UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Análisis, diseño e implementación de un sistema de recogida y visualización en tiempo real de elementos multimedia

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Pablo Alcarria Lozano

Tutores: Carlos Miguel Tavares De Araujo Cesariny Calafate
José María Cecília Canales

Curso 2020/2021

Resum

El projecte dut a terme consistix en l'anàlisi d'una aplicació d'una sola pàgina encarregada de visualitzar elements multimèdia de distintes xarxes socials per a l'ampliació d'esta. Es realitza un disseny i implementació d'un nou mòdul encarregat d'arreplegar les dades d'una nova font perquè es puguen visualitzar junt amb les altres fonts ja existents.

L'aplicació web arreplega distints temes socials, com la guerra, la política o catàstrofes mediambientals. Basant-se en estes preocupacions socials, l'aplicació permet la visualització en temps real de missatges i contingut multimèdia publicat en distintes xarxes socials, aplicant un processament del llenguatge natural als missatges arreplegats per a detectar les localitzacions, persones i organitzacions relacionades amb un tema.

Aquest mòdul afecta la part client i servidor de l'aplicació. En la part client s'explora el potencial oferit per React, una biblioteca de JavaScript, per a la construcció de pàgines web de forma dinàmica. En la part del servidor es crega un nou controlador encarregat de processar les peticions requerides pel client, realitzant consultes a un índex en Elasticsearch. Finalment, es realitzen proves per a verificar que les noves funcionalitats afegides a la pàgina satisfan els requisits identificats i el resultat obtingut concorda amb els objectius.

Paraules clau: JavaScript, React.JS, Full Stack, SPA, sistemes en temps real, aplicació web, API, REST, SCRUM, Elasticsearch

Resumen

El proyecto llevado a cabo consiste en el análisis de una aplicación de una sola página encargada de visualizar elementos multimedia de distintas redes sociales para la ampliación de ésta. Se realiza un diseño e implementación de un nuevo módulo encargado de recoger los datos de una nueva fuente para que se puedan visualizar junto a las demás fuentes ya existentes.

La aplicación web recoge distintos temas sociales, como la guerra, la política o catástrofes medioambientales. En base a estas preocupaciones sociales, la aplicación permite la visualización en tiempo real de mensajes y contenido multimedia publicado en distintas redes sociales, aplicando un procesamiento del lenguaje natural a los mensajes recogidos para detectar las localizaciones, personas y organizaciones relacionadas con un tema.

Este módulo afecta tanto la parte cliente de la aplicación como en la del servidor. En la parte cliente se explora el potencial ofrecido por React, una biblioteca de JavaScript, para la construcción de páginas web de forma dinámica. En el lado del servidor se crea un nuevo controlador encargado de procesar las peticiones requeridas por el cliente, realizando consultas a un índice en Elasticsearch. Por último, se realizan pruebas para verificar que las nuevas funcionalidades añadidas a la página satisfacen los requisitos identificados y el resultado obtenido concuerda con los objetivos.

Palabras clave: JavaScript, React.JS, Full Stack, SPA, sistemas en tiempo real, aplicación web, API, REST, SCRUM, Elasticsearch

Abstract

The project consists of the analysis of a single page application destined to displaying multimedia elements from different social networks for its extension. A new module it's designed and implemented to collect data from a new source in order to be displayed next to the other existing sources.

The web application collects different social issues, such as war, politics or environmental catastrophes. Based on these social concerns, the application enables real-time visualization of messages and multimedia content posted on different social networks, applying natural language processing to the collected messages to detect locations, people and organizations related to a topic.

This module affects both the client side of the application and the server side. On the client side It is explored the potential offered by React, a JavaScript library, for building web pages dynamically. On the server side, a new controller is created in charge of processing the requests required by the client, making queries to an index in Elasticsearch. Finally, tests are performed to verify that the new functionalities added to the page satisfy the identified requirements and the result obtained matches the objectives.

Key words: JavaScript, React.JS, Full Stack, SPA, real-time system, web application, API, REST, SCRUM, Elasticsearch

Índice general

Índice de figuras

CAPÍTULO 1
Introducción

1.1 Motivación

Las preocupaciones de la sociedad han sido clave en el desarrollo del ser humano desde antes de la irrupción de Internet de forma total en nuestras vidas. Por lo tanto, el estudio del comportamiento social de la población, el social sensing, es un paso lógico en un mundo cada vez más conectado y con más interacciones sociales que nunca. Gracias a internet, la información ha sido más accesible para gran parte de la población, y hay gente que lucha a diario para que pueda seguir siéndolo. Sin embargo, el contenido creado en Internet cada vez crece más, y las tendencias, temas y modas son pasajeras, ocupando gran parte del contenido que consumimos en nuestro día a día, y generando un ruido en la información que podemos obtener sobre nuestro entorno. Este contenido, cuya intención es únicamente entretener, quita el foco de los temas que realmente afectan a la población: su salud y el estado de bienestar. Contribuir al desarrollo un sistema que recoja el pensamiento colectivo y ayude a reflejar la sociedad y sus problemáticas me parece un uso responsable y necesario de las tecnologías disponibles.

Dado que durante el Grado en Ingeniería Informática me he especializado en la rama de las tecnologías de la comunicación e información, el desarrollo de este sistema es la oportunidad perfecta para demostrar los conocimientos y aptitudes obtenidas durante mi formación, además de demostrar el potencial de nuevas tecnologías que tienen una gran demanda en el mercado laboral.

1.2 Objetivos

El objetivo de este trabajo de fin de grado es desarrollar un sistema de recogida de recogida y visualización de elementos multimedia. Dado que el proyecto se desarrolla dentro de una aplicación ya existente, es también necesario realizar un análisis del funcionamiento actual de la aplicación y sus tecnologías. Para lograr este objetivo se definen los siguientes objetivos específicos:

- Analizar la estructura de una aplicación web de recolección de elementos multimedia en tiempo real basada en el modelo “Single Page Application”.

- Identificar los requisitos de la aplicación para realizar el desarrollo y crear, en base a ellos, un nuevo controlador para el contenido de los mensajes procesados de Youtube.

- Implementar un módulo de visualización de los comentarios de Youtube e integrar sus datos para su correcta visualización junto a las demás fuentes de datos ya integradas: Twitter, RSS y Telegram.

1.3 Estructura del documento

A continuación se detalla la estructura de esta memoria de TFG:

Capítulo II Se habla sobre el estado del arte en el social sensing junto al procesamiento natural del lenguaje, y sobre las distintas fuentes donde podemos encontrar los temas que más preocupan a la población.

Capítulo III Se explica la metodología ágil SCRUM aplicada durante el desarrollo del trabajo.

Capítulo IV Se describen las tecnologías empleadas en la aplicación web, tanto en la parte *front-end* como *back-end* .

Capítulo V Se describen los pasos relacionados con la implementación del trabajo realizado. Dentro de este apartado se detallan aspectos como el prototipado utilizado para la parte frontal, el diseño de la solución *back-end* , y cómo se visualizan los datos dentro de la aplicación.

Capítulo VI Se detallan las validaciones y pruebas seguidas para comprobar que el sistema es totalmente funcional y cumple los requisitos captados en la fase de análisis.

Capítulo VII Se reflejan las conclusiones obtenidas durante el desarrollo del proyecto, y se discuten posibles trabajos futuros dentro del proyecto, mejoras y funcionalidades añadidas.

CAPÍTULO 2
Estado del arte

Los humanos somos uno de los sensores más versátiles que tenemos a nuestra disposición para recoger y procesar datos. A diferencia de los sensores creados específicamente para desempeñar esa función, el ser humano, por defecto, tiene un conocimiento del contexto, detecta anomalías, e interpreta escenarios complejos de manera más sencilla que los sensores. Hay diversas formas en las que podemos estar involucrados en los sistemas de detección, o sensing. Wang, Dong et al. describen [1] tres formas de involucrarse:

- Fuentes de información. El ser humano es un gran observador. Siempre ha buscado de una forma u otra describir y comprender su entorno. Por ejemplo, en el campo de la inteligencia militar, se ha confiado en las fuentes humanas mucho antes de la introducción de sensores físicos. Los autores de "Social Sensing : Building Reliable Systems on Unreliable Data" explican que, en una conversación que tuvo uno de los escritores con un amigo, le confesó que, cuando siente un temblor en la tierra o algo que pueda ser parecido a un terremoto, primero acude a twitter [1]. Esto sería seguramente lo que haríamos muchos de nosotros: consultar a nuestros allegados.
- Operadores de sensores. El trabajo con sensores como cámaras o aplicaciones de geoetiquetado, donde la detección se realiza por el ser humano pero se lleva a cabo por un dispositivo. En este caso, se continúa aprovechando el conocimiento del entorno que puede tener un ser humano en un contexto, por lo que sabe qué observaciones y/o anomalías reportar.
- Portadores de sensores. Se puede contribuir como portador de sensores con los dispositivos que usamos a diario. Actualmente, nuestros smartphones y wearables como pulseras de actividad o relojes inteligentes, vienen equipados con una gran cantidad de sensores. La explotación de estos sensores es natural en aplicaciones que se centran en mediciones de espacios sociales.

Además de los papeles que pueden desempeñar los humanos en la detección, es interesante conocer cómo participan. Pueden participar activamente o compartir los datos por su cuenta, incentivados por el beneficio que reciben al compartirlo. Se explican también tres tipos de corrientes que podemos encontrar:

- Auge del uso de dispositivos de detección. Esta primera tendencia va ligada a la proliferación de los sistemas de detección disponibles para el consumidor. Sistemas de monitorización de actividad física, smartphones con cámara, acelerómetros, giroscopios y GPS, e incluso plataformas de entretenimiento como Wii-fit; ofrecen oportunidades para la recopilación de datos que además proporcionan un beneficio directo en el consumidor, ya que pueden observar sus resultados y conocer mejor sus propios hábitos.

- Conectividad móvil. Gracias al acceso ubicuo a Internet desde cualquier lugar somos capaces de medir y reportar eventos en tiempo real. A parte de las posibilidades que nos ofrecen los smartphones con su conectividad a Internet, puede destacar más su uso en aplicaciones que exploten la conectividad en otros ámbitos, como el tráfico de vehículos, desde coches hasta drones, o vehículos no tripulados. Las posibilidades que se ofrecen pueden estar más basadas en la recolección de datos y estadísticas para monitorizar el tráfico o para alertar de situaciones de emergencia o posibles accidentes.
- Redes sociales. Esta tercera corriente, la cual ha sido clave en el campo de la detección social, es la popularidad de las redes sociales esta última década. Estos portales ofrecen canales de difusión de información de forma masiva, donde las noticias y sucesos se pueden transmitir rápidamente en tiempo real. La irrupción de estas redes da la oportunidad de crear aplicaciones que recopilen y compartan este contenido de una manera lógica y organizada.

Centrándonos en las aplicaciones de sensing, se detallan tres tipos dependiendo de su funcionalidad:

- Aplicaciones centradas en puntos de datos. Los usuarios comparten puntos de datos (sus observaciones) individualmente, que son disponibles para los clientes o responsables de la toma de decisiones, con la intención de ayudar en la toma de estas. Un ejemplo podría ser una aplicación de geoetiquetado donde los individuos comparten imágenes geolocalizadas relevantes para la aplicación en cuestión. Por ejemplo, para documentar la calidad de las carreteras [2] o las localizaciones de contenedores dentro de una infraestructura [3].
- Aplicaciones centradas en las estadísticas. Se calculan estadísticas como promedios, probabilidades o distribuciones, extraídas de los datos recibidos. Las aplicaciones son muchas, desde calcular el tráfico en una ciudad para evitar atascos, como con CarTel [4], hasta medir la actividad física y los patrones de comportamiento con HealthAware [5]. También nos encontramos la propuesta de BikeNet [6], que trabaja con sensores físicos para recoger datos de los ciclistas para mejorar sus actividades. Tan [7] nos propone un robot autónomo subacuático con sensores que recoja información para monitorizar la calidad de las aguas de lagos.
- Aplicaciones centradas en modelos de datos: Los modelos se crean en base a la recogida de datos sensoriales, que pueden utilizarse para la toma de decisiones fuera de la ubicación de la recolección de datos. Gracias a la irrupción del Internet de las cosas, este tercer tipo puede recoger datos de una gran variedad de fuentes. Por ejemplo, sensores que detallen el consumo eléctrico de una casa pueden ayudarnos a aumentar la eficiencia energética; o los datos recogidos por sensores de actividad de usuarios corrientes y entusiastas del deporte puede contribuir a la creación de campañas que promuevan actividades saludables para la población [8].

En la recogida de datos escritos, como mensajes, noticias o comentarios, es necesario poner el foco de la atención en el procesamiento natural del lenguaje. Todos los mensajes recogidos crean una nube que es necesaria clasificar, determinar su validez e impacto. Podemos encontrar ejemplos de este trabajo en el TASS, un taller que lleva más de siete ediciones donde se analizan los sentimientos de mensajes de Twitter, de forma que estos se clasifican en base a la positividad, negatividad o neutralidad del mensaje [9]. También tenemos ocasionalmente capas de complejidad, con mensajes irónicos o cómicos [10]. Las empresas ya son conocedoras de la importancia de la percepción de los clientes sobre su

producto, así que incluso buscadores como Bing ya incluyen el análisis de sentimientos en sus búsquedas [11]. Los mensajes relativos a un tópico, organización o acontecimiento son realmente útiles para conocer la opinión al respecto. Dentro del ámbito empresarial, el procesamiento de estos mensajes puede ayudar a conocer el impacto de sus productos en los usuarios.

Centrándonos el impacto de las redes sociales en la detección, el uso de ellas de forma aislada no parece una alternativa sólida. Las redes sociales como Twitter, Instagram, Facebook o YouTube, muestran información relevante en base a las preferencias del usuario que van recogiéndose a medida que el usuario interactúa con la página y consume el contenido que se le muestra. Así se establece qué información y contenido es relevante para el usuario, y muestra un contenido relacionado en base a eso, además del contenido de las personas a las que sigue. Las cuentas a las que sigue un usuario va a condicionar la información recibida, además de crear comunidades sesgadas ideológicamente dentro de las redes sociales [12]. La cobertura informativa de problemas, como el confinamiento por el Covid-19, puede acentuar la creación de este tipo de grupos en los que no se obtiene una información contrastada o ausente de sensacionalismos [13].

Un apartado relevante para la información en tiempo real del entorno es el de las tendencias, donde se puede ver cuáles son los eventos más mediáticos que han sucedido recientemente, o los que están sucediendo en el momento que se explora, pero tiene dos debilidades principales. En primer lugar, los temas mostrados en las tendencias pueden comprarse o crearse de forma engañosa. Las empresas pueden patrocinar un "hashtag" para promocionar un nuevo producto, un evento, o cualquier contenido que quieran posicionar, contenido que por lo general no guarda relación con ningún tema social que tenga un impacto en la sociedad, y que pueda condicionar o impactar en el día a día. En segundo lugar, existe la posibilidad de posicionar como tendencia un tema de forma artificial en base a los intereses de organizaciones o grupos. Con la ayuda del uso de programas informáticos o bots que manejen una gran cantidad de cuentas, en la mayoría de ocasiones falsas y dirigidas por un conjunto de personas, y mediante la publicación de una gran cantidad mensajes con esas cuentas, pueden llegar a posicionar el tema, empresa u organización de la que hablen como relevante, y que la red social en cuestión lo posicione como tendencia. Esto es peligroso ya que puede crear campañas de desinformación u odio.

Por otra parte, en España tenemos el trabajo del Centro de Investigaciones Sociológicas (CIS), un organismo autónomo cuya finalidad es el estudio de la sociedad española. Se encarga de elaborar encuestas periódicamente y difunde abiertamente los datos de estas encuestas. Conllevan un gran trabajo, tanto realizarlas como luego procesarlas. Esto hace que los datos obtenidos no sean en tiempo real, pero son relevantes en el momento de la publicación, así que en base a sus estudios es posible determinar qué preocupaciones existen en la población de forma eficaz [14]. Entre los estudios que realiza el CIS, nos encontramos con los barómetros, que son encuestas mensuales, encuestas monográficas que comprenden un amplio abanico de temas, o los indicadores de confianza de los consumidores (ICC).

CAPÍTULO 3
Metodología

Para el desarrollo del proyecto se ha seguido la metodología ágil e incremental SCRUM [15]. Esta metodología consiste en la división del proyecto en distintas etapas llamadas **sprints**, en las que al final de éstas puede haber un entregable. Dentro de SCRUM se definen tres tipos de roles:

1. Product Owner (PO). Responsable del proyecto y de representar los intereses de los stakeholders. Debe encargarse de gestionar qué actividades se deben priorizar en el desarrollo del producto. Las necesidades identificadas deben aplicarse para crear historias de usuario, que se utilizarán para definir los entregables del proyecto.

2. Scrum Master (SM). Encargado de organizar las tareas dentro del Development Team y asegurarse de que se apliquen las metodologías ágiles correctamente. Actúa como intermediario entre el Product Owner y el Development Team, moderando las reuniones realizadas y revisando el product backlog.

3. Development Team (DT). Consiste en un equipo de 5 personas encargadas de la implementación del proyecto. Son los encargados de realizar las tareas asignadas en cada sprint. En este rol se incluye el trabajo recogido en esta memoria.

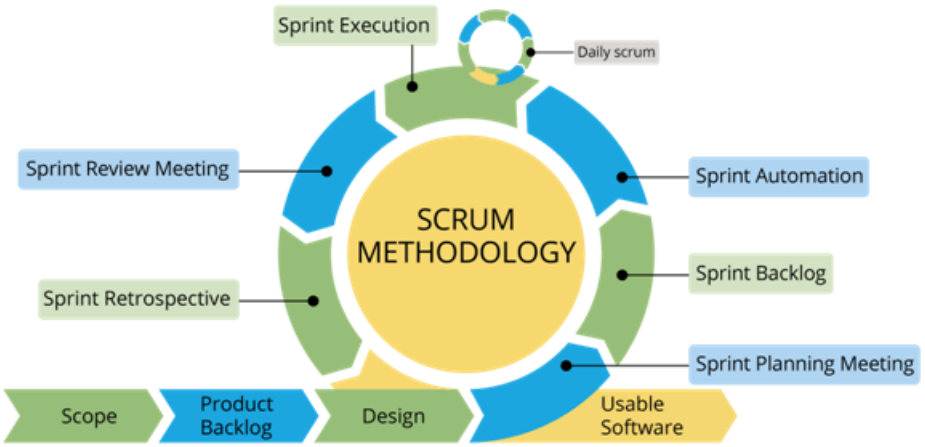

Figura 3.1: Ciclo del desarrollo mediante SCRUM.
Fuente: https://blog.wanchingteoh.me/project-management-why-agile-scrum-ckmuov53y00mt5us10wr06cow

El tiempo asignado a cada sprint es de unos 10 días de trabajo o 2 semanas, con reuniones periódicas al finalizar cada sprint. En total se han realizado 6 sprints. Cada

sprint está constituido de varias fases, comenzando por la reunión de planificación, donde se reúne el equipo para seleccionar las tareas que deben realizarse entre todas las existentes en el *backlog* atendiendo a las necesidades del proyecto y su prioridad, así como añadir nuevas tareas en este *backlog*. Por ello, estas reuniones proporcionan un desarrollo más flexible y adecuado a las características del proyecto. Una vez determinadas las tareas dentro de un sprint, durante la ejecución del mismo existe una breve reunión diaria, llamada *daily*, donde cada miembro del equipo explica lo que se ha trabajado el anterior día y expresa las dudas, decisiones tomadas, y los problemas encontrados (si existiesen). Estas reuniones en las que participa el DT deben ser breves, de aproximadamente quince minutos, y produciéndose en la misma hora y lugar siempre para reducir la complejidad. En el caso de que el PO o el SM estuviesen trabajando activamente en alguna de las tareas del sprint, también participan en esta reunión como desarrolladores. Una vez terminada la ejecución del sprint, el equipo vuelve a realizar una reunión llamada *review* para hacer retrospectiva de la ejecución del sprint, donde se comprueba si esta ejecución ha sido llevada a cabo con éxito. Esta metodología iterativa proporciona reuniones periódicas que permiten analizar y testear las nuevas funcionalidades creadas en cada *sprint*, así como identificar nuevos requisitos de forma temprana o redirigir los siguientes *sprints* para adecuarlo a las necesidades del PO.

Como se puede apreciar por las características de la metodología ágil, ésta es más efectiva en equipos pequeños de menos de diez personas aproximadamente, dado que la ejecución de las tareas puede variar de forma continua, y se realiza un gran número de reuniones. Cuanto más grande es el equipo, más personas deben organizarse, por lo que en un equipo con mucha gente trabajando puede llegar un punto en el que se dedica demasiado tiempo a la organización del proyecto.

A continuación se resume el trabajo hecho en cada sprint, que se detallará en la fase de implementación:

- Sprint 0. Se instala el software necesario y de la versión de desarrollo del proyecto en cuestión.
- Sprint 1. Se realiza el diseño del prototipado, y se identifican los componentes React que van a tener que modificarse, así como los componentes que deberán ser implementados. Se obtiene acceso a la API y se configura correctamente para realizar el desarrollo. En la siguiente reunión se comentan las propuestas de prototipado entregadas.
- Sprint 2. En base a una maquetación de los datos que se recibirán desde la API, se implementa la visualización de estos datos en el front-end, desarrollando los componentes identificados en el sprint anterior.
- Sprint 3. Se crea un *endpoint* dentro de la API que realice las consultas a la base de datos, y se comprueba que estos datos se reciben correctamente en el front-end.
- Sprint 4. Los datos recogidos se añaden en las vistas globales de la aplicación, así como a las gráficas globales.
- Sprint 5. Se manejan las peticiones que añaden los datos a las gráficas de visualización del tráfico de mensajes, recuento de palabras, y localización de los mensajes.
- Sprint 6. Se realizan pruebas de la aplicación, y se confirma de que el comportamiento de la aplicación es el esperado tras el desarrollo añadido.

Durante el transcurso del proyecto se realizan periódicamente reuniones para determinar cada sprint. En estas reuniones el proceso de definición seguido para cada sprint es el siguiente:

1. Se definen los requisitos que deben cumplirse mediante historias de usuario.
2. Se asigna el trabajo a realizar por cada integrante del grupo dentro del sprint.
3. Se identifican las posibles dependencias que puedan existir para estas tareas.
4. Se calcula el tiempo estimado para realizar la tarea mediante la técnica de planning poker.

Para organizar las actividades de los sprints se utiliza el tablero Kanban. Este sistema de trabajo consiste en un tablero de trabajo en el que se añaden las tareas identificadas, las tareas que se están llevando a cabo en el sprint, y las tareas ya terminadas. Con el tablero Kanban se obtiene una mejor visualización del flujo de trabajo, y del estado de cada sprint. Con esta metodología se consigue delimitar, dentro de la producción, las diferentes fases de la ejecución. Como software para el tablero Kanban se utiliza Trello, una herramienta en línea colaborativa en la que cada integrante del equipo puede realizar modificaciones, añadir comentarios, y notificar al resto de integrantes en qué estado se encuentra su trabajo. Un ejemplo de tablero se puede observar en la figura 3.2, donde el contenido está organizado en cuatro columnas con tarjetas, que corresponden a las diferentes tareas; estas son: backlog, para hacer, en proceso, y hecho.

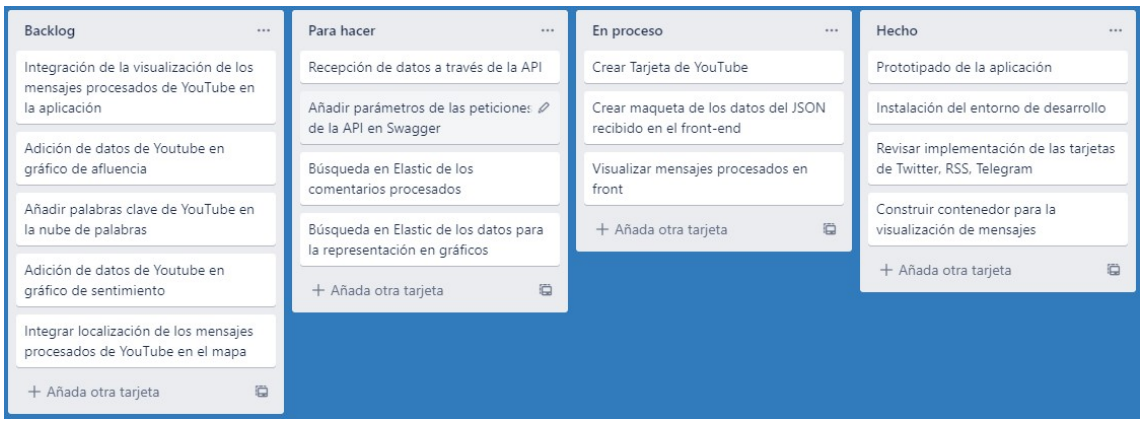

Figura 3.2: Tablero Kanban del proyecto.

Esta metodología, según los creadores y autores de la guía más actualizada [15], para su éxito y su correcta aplicación, depende de la competencia del equipo en cinco aspectos clave:

1. Compromiso: el equipo se compromete a lograr sus objetivos y apoyarse mutuamente.
2. Enfoque: el enfoque principal del sprint es realizar el mejor progreso posible hacia los objetivos marcados.
3. Apertura: las partes interesadas en el proyecto deben tener una mentalidad abierta respecto al trabajo y los objetivos, dado que pueden realizarse cambios.
4. Respeto: los integrantes se respetan entre sí para ser personas capaces e independientes, y son respetados como tales por las personas con las que trabajan conjuntamente.
5. Valor: los miembros tienen el valor de hacer lo correcto y trabajar en la resolución de los problemas.

CAPÍTULO 4
Visión general de la aplicación de referencia

En el siguiente capítulo se detallan las distintas tecnologías utilizadas en la aplicación web en el comienzo del trabajo. Podemos encontrar dos partes totalmente delimitadas: la parte del servidor y la parte del cliente. Encontramos el uso de JavaScript tanto en el lado del servidor como en el cliente, por lo que el uso de NPM para la gestión de paquetes y la ejecución es común en ambas partes.

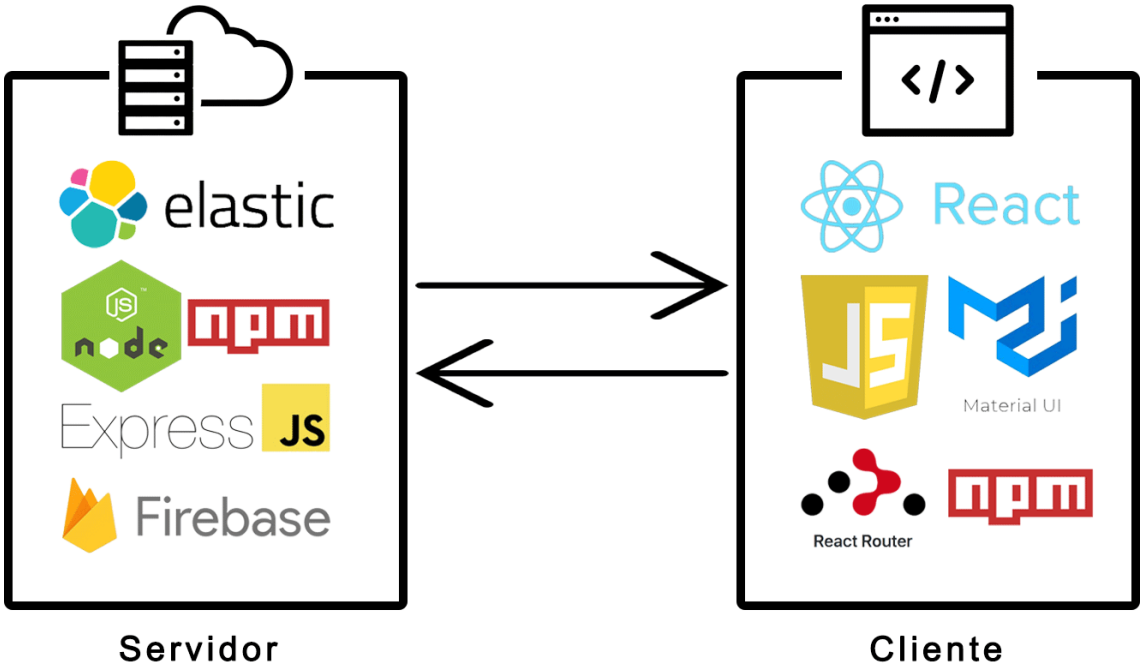

Figura 4.1: Esquema de las tecnologías más relevantes en el servidor y el cliente.

En el lado del cliente nos encontramos con una aplicación basada en la arquitectura de una única página, donde destaca el uso de la librería React, basada en el lenguaje de programación JavaScript. En la estructura de los documentos de la aplicación no nos encontramos con ningún archivo HTML, dado que React permite la creación de HTML en sus archivos JavaScript. En su lugar nos encontramos con un gran conjunto de componentes React, en formato JSX. Dentro contienen JavaScript, HTML y CSS en un mismo documento, que en su conjunto crean las vistas de la aplicación con los estilos ya aplica-

dos. Cabe destacar que los estilos de estos componentes, escritos en CSS, pueden estar en documentos a parte del archivo en el que se crea el componente, pero gracias al uso de Material UI disponemos de la función "withStyles", que aplica los estilos almacenados en una variable del propio archivo en el renderizado de la aplicación.

En el otro lado, en la parte del servidor, también nos encontramos el uso de JavaScript junto a Node.JS para la gestión de las comunicaciones asíncronas. Para el manejo de las peticiones de la API basada en el diseño REST se emplea React.JS, que permite escribir respuestas a URL específicas, por lo que se complementa con el uso de React Route en la parte del cliente. El contenido que devuelven estas peticiones son los mensajes procesados y datos relacionados a ellos, que están alojados en Elasticsearch después de ser recolectados por un *crawler*, y procesados haciendo un análisis del sentimiento de cada mensaje. Además, se explica en detalle la API creada en el servidor que emplea las tecnologías mencionadas, donde se implementa un controlador durante el desarrollo del proyecto.

Por último, por las tecnologías empleadas en la parte frontal, nos encontramos con una aplicación web que sigue la arquitectura de una aplicación de una única página, por lo que se explican sus características y las principales diferencias respecto a la arquitectura web tradicional multi-página.

4.1 Tecnologías empleadas

A continuación se detallan las tecnologías más relevantes utilizadas durante el desarrollo del proyecto, además de mencionar el entorno en el que se ha desarrollado el trabajo.

4.1.1. JavaScript

JavaScript es un lenguaje de programación de alto nivel, interpretado, orientado a objetos, y con un tipado débil. Está soportado por los navegadores modernos, tanto de escritorio como móviles. Es usado tanto en la parte cliente como servidor, siendo más visible la parte del cliente por su interacción con el DOM (Modelo de Objetos del Documento) en la parte *front-end* de las páginas web.

4.1.2. Node.JS

Construido sobre el motor de JavaScript, es un entorno de ejecución orientado a eventos asíncronos que permite utilizar JavaScript en la parte del servidor. Con Node.JS nos encargamos de la parte de la comunicación bidireccional en tiempo real entre el cliente y el servidor. Gracias a su manejo de eventos, podemos crear aplicaciones en red rápidamente con una alta disponibilidad, dado que los procesos difícilmente se bloquean [16].

4.1.3. React

También conocido como ReactJS, es una biblioteca JavaScript de código abierto desarrollada por Facebook, y usada principalmente para crear interfaces de usuario siguiendo una arquitectura SPA (Single Page Application). React utiliza un DOM virtual que compara con el original para detectar los cambios y realizar las actualizaciones en la aplicación, en lugar de sólo utilizar el DOM del navegador. En React, las páginas web se crean con componentes, que son módulos aislados que, en conjunto, crean la página web con

todas sus funcionalidades y vistas. Esto facilita la reutilización de los componentes y nos permite diferenciarlos entre ellos. Es utilizada en grandes páginas como Imgur o Airbnb.

React utiliza JSX, que es una extensión de JavaScript que nos permite escribir código XML y HTML dentro de JavaScript. Usarlo es opcional, pero es muy útil en la creación de componentes ya que une la lógica de renderizado con la lógica de la interfaz de usuario en los componentes, y estos componentes son los que separan los intereses dentro de una página. Además, a la hora de renderizar el contenido en la página, ayuda a prevenir vulnerabilidades XSS [17].

El DOM Virtual

Para comprender cómo se actualizan los componentes dentro de una página, primero es necesario entender el Modelo de Objetos del Documento, conocido como DOM. Es una interfaz para representar los documentos HTML y XML que define cómo poder acceder al contenido de estos documentos para modificar su estructura, estilo o su contenido. El responsable del DOM y su especificación es el World Wide Web Consortium (W3C). El contenido del documento se representa con nodos y elementos que tienen métodos y propiedades, y permite el acceso dinámico a través de la programación para modificar su estructura con lenguajes como ECMAScript.

Uno de los lenguajes de programación más utilizados para interactuar con el DOM es JavaScript. La forma de acceder a los distintos componentes de una página web mediante JavaScript es a través del objeto document, que representa el árbol de la página, donde está su contenido. El DOM no es parte de JavaScript, sino que JavaScript trabaja sobre el DOM.

ReactJS utiliza un DOM virtual, que no es más que una réplica en JavaScript del DOM cargado en memoria. Esta copia se compara con el DOM no virtual, de forma que, en el momento en el que se detecta una diferencia en el estado de cualquier componente entre los dos DOM, la página vuelve a renderizar los componentes afectados sin necesidad de volver a recargar la página.

Librerías usadas en React

Al ser una biblioteca de JavaScript, React tiene compatibilidad con una gran variedad de plug-ins para aprovechar más sus ventajas de desarrollar sobre ella. A continuación se describen las más relevantes utilizadas en el proyecto:

React Route

Biblioteca para React utilizada para manejar el enrutamiento dentro del lado del cliente de la aplicación web, facilitando la navegación gracias al uso de componentes para definir las rutas. Route, a partir de la versión 4, nos ayuda a gestionar el renderizado de los componentes de React gracias a su enrutamiento dinámico. Con el uso de esta librería, la creación de rutas dentro de las aplicaciones SPA es más fácil de gestionar, ya que en una aplicación de una única página no disponemos de distintos documentos HTML independientes que estén diferenciados entre sí. En un proyecto multi-página de una aplicación web tenemos un documento HTML para cada vista de la página, pero, en el caso de las SPA, sólo se requiere una única página que va renderizando el contenido dinámicamente. Con esta biblioteca podremos crear las rutas necesarias como si estuviésemos en una aplicación web con múltiples páginas. Por lo tanto, el uso de React Route es muy recomendable para mantener un orden dentro de la estructura del proyecto.

Viéndolo en un ejemplo: si en una página web con un modelo tradicional, en el que tenemos para cada página un documento HTML asociado, por ejemplo una página dedicada a la información de contacto sea la dirección "www.dominio.es/contacto", en React Route podemos crear un componente que, como valor de ruta en sus propiedades, tenga "/contacto".

Highcharts

Para la visualización de los datos, la aplicación utiliza la librería Highcharts. Es una librería escrita en JavaScript la cual tiene integración completa con React, permitiendo su actualización en tiempo real, así que es perfecta para el proyecto que se realiza. Es utilizada por grandes empresas como Facebook, Twitter, Ericsson o MasterCard. Tiene una licencia no comercial para organizaciones sin ánimo de lucro. Además, el estilo puede ser modificado vía JavaScript o CSS, por lo que, dentro de un proyecto basado en una librería de JavaScript, no deben presentarse complicaciones para modificar el aspecto de las tablas para que cumplan los requisitos de nuestra aplicación.

Material UI

Para el diseño de la aplicación se utiliza Material UI, un set de componentes para React que sigue la normativa de diseño Material Design. Esta normativa fue desarrollada por Google e implementada tanto en dispositivos Android como en la web. Proporciona un diseño unificado y responsive que utiliza un *grid* de 12 columnas, al igual que Bootstrap.

4.1.4. Bitbucket

Es un servicio de alojamiento de proyectos que utiliza el sistema de control de versiones GIT. En Visual Studio Code disponemos de una extensión para su uso dentro del editor, lo cual nos permite clonar los repositorios desde el propio editor.

4.1.5. ElasticSearch

Es un motor de analítica y análisis distribuido que ha sido desarrollado a partir de Apache Lucene. Destaca por su velocidad y capacidad de indexar muchos tipos de contenido, como análisis y visualización de datos. Elasticsearch indexa los datos, almacenados en formato JSON, y los usuarios pueden ejecutar consultas sobre esos datos.

4.1.6. Firebase

Proporciona el alojamiento de la aplicación. Es una plataforma de Google creada para el desarrollo de aplicaciones web y móviles, alojada en la nube. Dispone de herramientas de autenticación de usuarios, almacenamiento en la nube, y sincronización de datos en los proyectos sin una lógica de sincronización compleja.

4.1.7. NPM

Node Package Manager es un gestor de paquetes encargado de añadir fácilmente los módulos que sean necesarios a lo largo del desarrollo del un proyecto. Además de la

ayuda que proporciona con la gestión de librerías y dependencias dentro de la aplicación, también permite el uso de scripts para definir tareas dentro del proyecto, como por ejemplo para la ejecución del mismo de forma local en el entorno de desarrollo.

4.1.8. Express.JS

Librería utilizada para crear infraestructuras de aplicaciones web sobre NodeJS. Su uso acelera la creación de APIs basadas en diseño REST por la facilidad con la que se pueden controlar las peticiones HTTP. Para manejar las diversas peticiones que existen en cualquier aplicación web, incluye un manejo de rutas con las que se asocia a cada ruta una función, que es encargada de procesar la petición correspondiente.

4.1.9. Entorno de desarrollo

Para el desarrollo se ha hecho uso del editor de código de uso libre Visual Studio Code, disponible para Linux, MacOS y Windows, por lo que es una buena opción dentro de equipos de desarrollo donde es posible que se esté trabajando desde distintos sistemas operativos en remoto. Permite realizar todas las tareas en un mismo entorno gráfico, y tiene las características suficientes para el trabajo que se va a desarrollar. Las principales fortalezas para escoger este editor de código son las siguientes:

- Resalta la sintaxis y autocompleta el código de JavaScript, React, CSS y HTML.
- Dispone de una gran variedad de extensiones que nos permiten trabajar de forma más productiva. Por ejemplo, extensiones como la de BitBucket nos ayuda al control de versiones, 'merges', 'commits' y 'pulls', o plantillas para crear métodos de una gran variedad de lenguajes de programación rápidamente.
- Dispone de una terminal integrada dentro del editor, por lo que dentro de de la aplicación podemos ejecutar el servidor local de pruebas, realizar conexiones ssh y utilizar utilidades del sistema.

4.2 API Utilizada

En la parte del *back-end* de la aplicación se trabaja con Node.JS. Se encarga de recibir las peticiones del *front-end* y hacer las consultas a la base de datos en Elasticsearch. En la aplicación ya existen tres fuentes de las que se reciben datos - Twitter, Telegram y noticias -, por lo que existen ya controladores de estas peticiones, además de otras para procesar las categorías dentro de la página web, la galería de imágenes, o la galería de vídeos.

El diseño de esta API web está basado en el modelo REST basado en el protocolo HTTP. Dentro de los *endpoint* que procesan las peticiones, como la solución requerida sólo recibe datos del *back-end* y no se contempla realizar un cambio de estado dentro del servidor, las peticiones se realizan mediante el método GET, devolviendo un código de estado 200 en el caso de procesar bien la petición, o un código de estado 400 si no se hace correctamente la petición. En cada *endpoint* de cada fuente se procesan dos peticiones que realizan consultas a Elasticsearch, una para los mensajes procesados y otra para la comparación de la afluencia de mensajes respecto a los sensores físicos. Ambas reciben los parámetros de la consulta en las peticiones, como se puede observar en la vista de Swagger de la figura 4.3. Gracias al uso de Express, la gestión de las peticiones se gestiona de manera sencilla. Las respuestas a las peticiones se devuelven en objetos JSON que contienen los datos resultantes de la búsqueda en la base de datos.

Gracias al uso de JavaScript y el formato JSON, tanto en la parte del servidor como en la parte cliente, el proceso del envío y recepción de datos se realiza de forma más ágil.

Respecto a las consultas realizadas sobre Elasticsearch, al tratarse de un motor de búsqueda no relacional, las peticiones no se realizan con lenguaje SQL, aunque Elastic también lo permite. Los datos están organizados dentro de índices en los que las consultas se estructuran con notación JSON. En las búsquedas, como se puede apreciar en la figura 4.2, se utilizan parámetros enviados en la petición. Dichos parámetros son esenciales para la construcción de la consulta, ya que en base a ellos se les da el valor adecuado. Como podemos observar en el siguiente ejemplo de una búsqueda de los mensajes procesados de Twitter, se crea una variable que sigue el formato JSON, que luego es enviada a la búsqueda en el índice de Elasticsearch:

```
var elasticsearchQuery = {
  "from": req.query.from,
  "size": req.query.size,
  "track_total_hits": true,
  "query": {
    "bool": {
      "must": [
        {
          "bool": {
            "should": categoryIdQuery
          }
        },
        {
          "range": {
            "tweet.created_at_timestamp": {
              "lte": req.params.newestTimestamp,
              "gte": req.params.oldestTimestamp
            }
          }
        }
      ]
    }
  },
  "sort": [
    {
      [orderPath]: {
        "order": req.query.orderType
      }
    }
  ]
}
```

Figura 4.2: Búsqueda de mensajes procesados de Twitter en un índice de Elasticsearch.

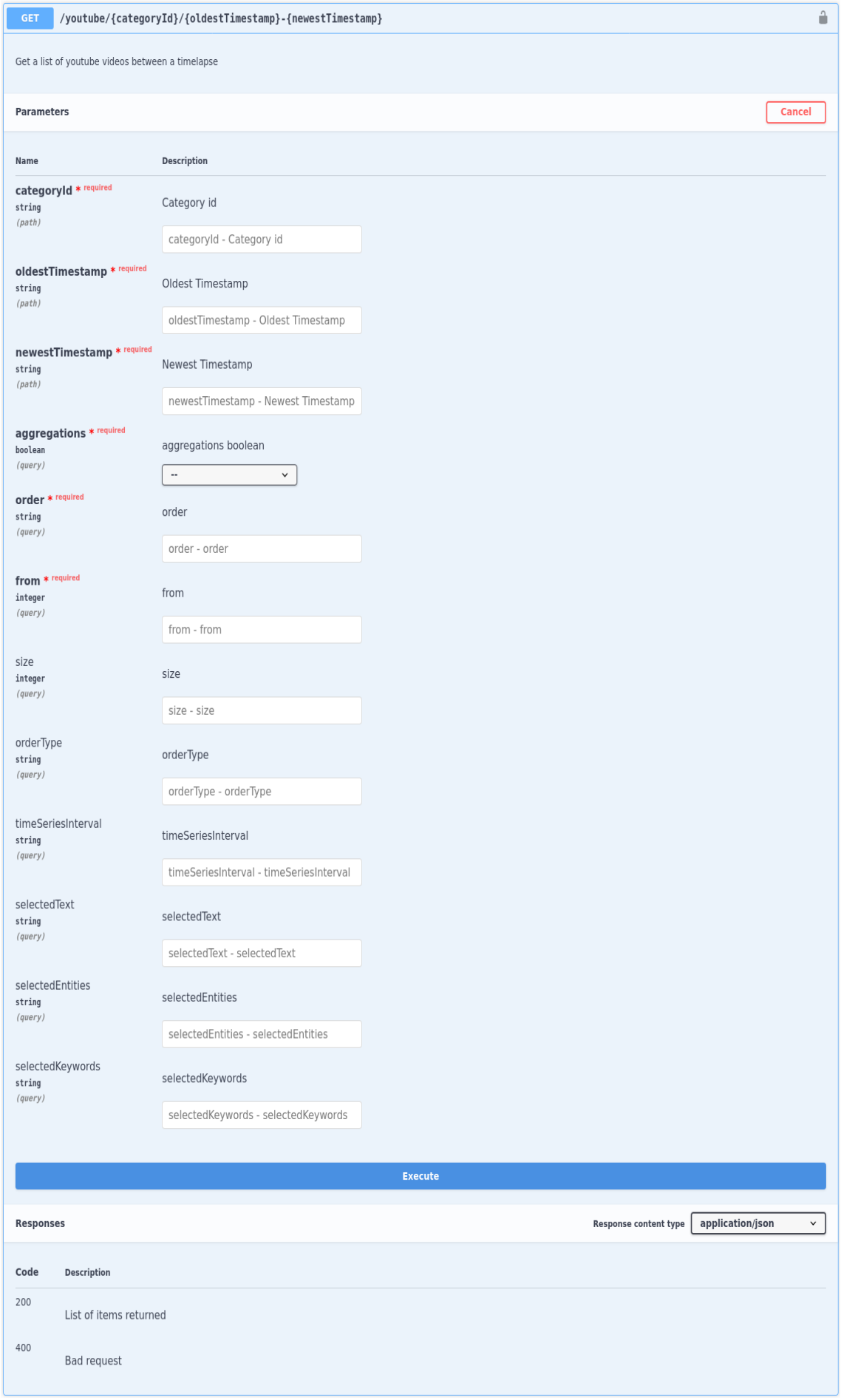

Figura 4.3: Swagger: detalles de un *endpoint* creado en la API.

4.3 Arquitectura de las Single-Page Application

Una Single-Page Application, o aplicación de página en castellano, es un sitio web que contiene todo el contenido en una única página con la intención de dar una experiencia más fluida y un menor tiempo de respuesta. Este tipo de aplicaciones tienen una complejidad mayor debido a su estructura modular, y que el contenido de la página cambia dinámicamente en función a la interacción del usuario.

Profundizando en la estructura modular, esta está dada por la división del contenido de la aplicación en componentes, que son los módulos con el contenido de la página. El uso de frameworks o librerías como React JS nos permite delimitar por completo cada componente de la aplicación, y nos ayuda a que sólo se actualicen los componentes afectados por la interacción del usuario. En conjunto, los componentes crean la aplicación, y, al ser actualizados y/o reemplazados independientemente, la página no necesita recargarse por completo por la interacción del usuario, lo cual conlleva una mejora en el tiempo de respuesta de la aplicación y un mayor rendimiento. En el caso de una aplicación construida sobre ReactJS y su DOM virtual, en el momento en el que el usuario interacciona con la página y se detecta un cambio en esta copia virtual del modelo de objetos del documento, los componentes afectados pueden ser reemplazados o actualizados rápidamente.

El hecho de tener una única página para toda la aplicación supone que en el modelo SPA no existan distintas páginas con contenido, sino diferentes vistas, que son los distintos apartados dentro de una web. Aunque sea la misma página, la URL puede cambiar a medida que se va navegando entre las vistas. Esto, además de ayudar al enrutamiento dentro de una aplicación, facilita la navegación por la página, y ayuda al usuario a saber dónde se ubica la vista que visualiza en cada momento.

Podría decirse que con las SPA se busca la experiencia de usuario de una aplicación de escritorio instalada en el ordenador del cliente, pero dentro de un navegador. Sin embargo, al acceder desde el navegador y no estar instalado, carece de desventajas como la instalación de plugins o las actualizaciones, y tiene las ventajas de no necesitar ningún entorno adicional, así como su rápida respuesta. Las SPA convierten el navegador en la interfaz de la aplicación, con una integración total entre todos los componentes de la aplicación web, y dándonos la posibilidad de visualizar contenido en tiempo real sin actualizar la página.

Comparando las ”Single-Page Application” con las ”Multi-Page Application”, también conocidas como “page redraw”, que es el modelo clásico de una página web en la que, para cada página solicitada, se crea una petición al servidor, éste devuelve una respuesta, y se vuelve a descargar todo el contenido, nos encontramos con un enfoque más moderno y que aprovecha mejor las tecnologías en el lado del cliente. La principal desventaja de las MPA la encontramos en la necesidad de recargar toda la página para visualizar el nuevo contenido. Además de la mejora del tiempo de respuesta que conseguimos con las SPA, se consigue una experiencia de usuario mejor, ya que no tienen que esperar a cargar el contenido. Por lo general, los hábitos de los clientes indican que cada vez tienen una menor tolerancia a la espera, y se intenta minimizar el tiempo de respuesta para no perder la atención de los usuarios. El hecho de no existir la necesidad de actualizar toda la página desde el lado del usuario supone una gran mejora respecto al modelo tradicional multi-página. Aún así, dentro de una MPA, sigue siendo posible el uso de peticiones AJAX también para mantener una comunicación asíncrona con la página web, siendo posible interactuar con el servidor sin necesidad de descargar de nuevo la página, y mejorando la interacción con ella.

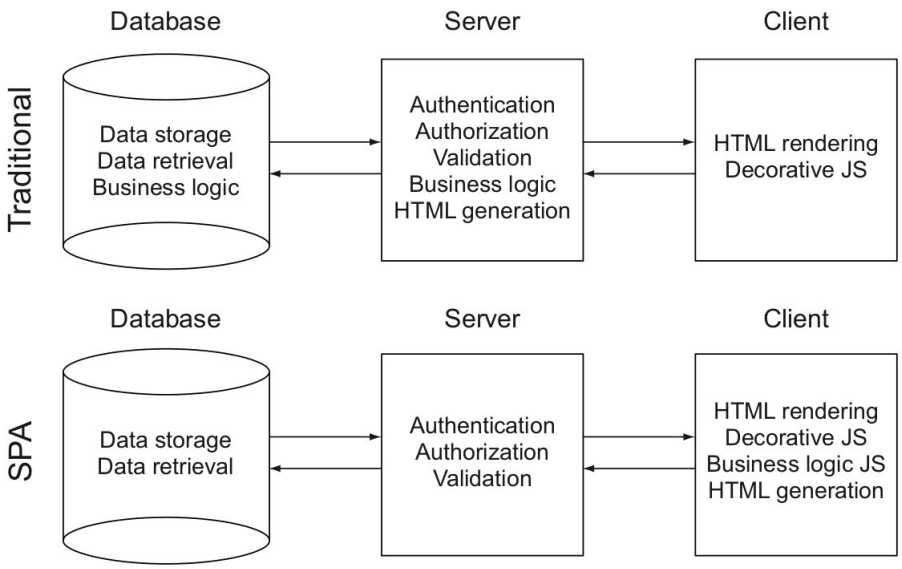

Figura 4.4: Comparación de la arquitectura de una página web tradicional y una SPA.

Otro beneficio del uso de componentes React en las SPA es que el mantenimiento y la gestión del código que hay detrás de dichos componentes es más fácil de mantener, ya que la implementación de cada parte está dentro de los módulos involucrados, y estos están bien delimitados dentro de la estructura del proyecto. Esto significa que, si un componente fuese cambiado o temporalmente desactivado, el resto de componentes independientes del modificado deberían ser totalmente funcionales. Trabajar con componentes en vez de con páginas web enteras permite la reutilización de partes comunes en diversas vistas de la aplicación. Por ejemplo, componentes que como la cabecera y el pie de la página, o componentes que sirven como contenedor para visualizar contenido dentro de éste. Con modificar el componente en cuestión tendremos todas las páginas actualizadas sin necesidad de modificar en todos los documentos HTML el contenido, y aumentando la productividad en el desarrollo ya que no es necesario volver a construir la página web, sino que se puede renderizar un componente ya creado anteriormente.

En esta arquitectura hay parte de la funcionalidad que se transfiere al cliente. También delega la lógica de JavaScript y la generación del HTML, ya que la mayoría de las veces suelen ir unidas, por lo que se requiere de más recursos por parte del cliente para ejecutar la página. Además, el código JavaScript que renderiza la página es visible desde el navegador, por lo que si se quiere mejorar la seguridad se debería trabajar con ofuscadores de código.

CAPÍTULO 5
Implementación

En este capítulo se aborda todo lo relacionado con el desarrollo del trabajo en la fase de desarrollo de la aplicación. En primer lugar se detalla el prototipado del que se parte, siguiendo con el diseño de la solución donde se detallan los elementos que deben crearse dentro de la aplicación, así como los ya existentes en la aplicación que son afectados; por último se incluyen los detalles de la implementación en cada sprint del desarrollo. Las tecnologías empleadas durante esta fase son las mismas que las descritas en el capítulo 4.1, siendo una gran ventaja el usar el mismo lenguaje para el *front-end* y el *back-end* .

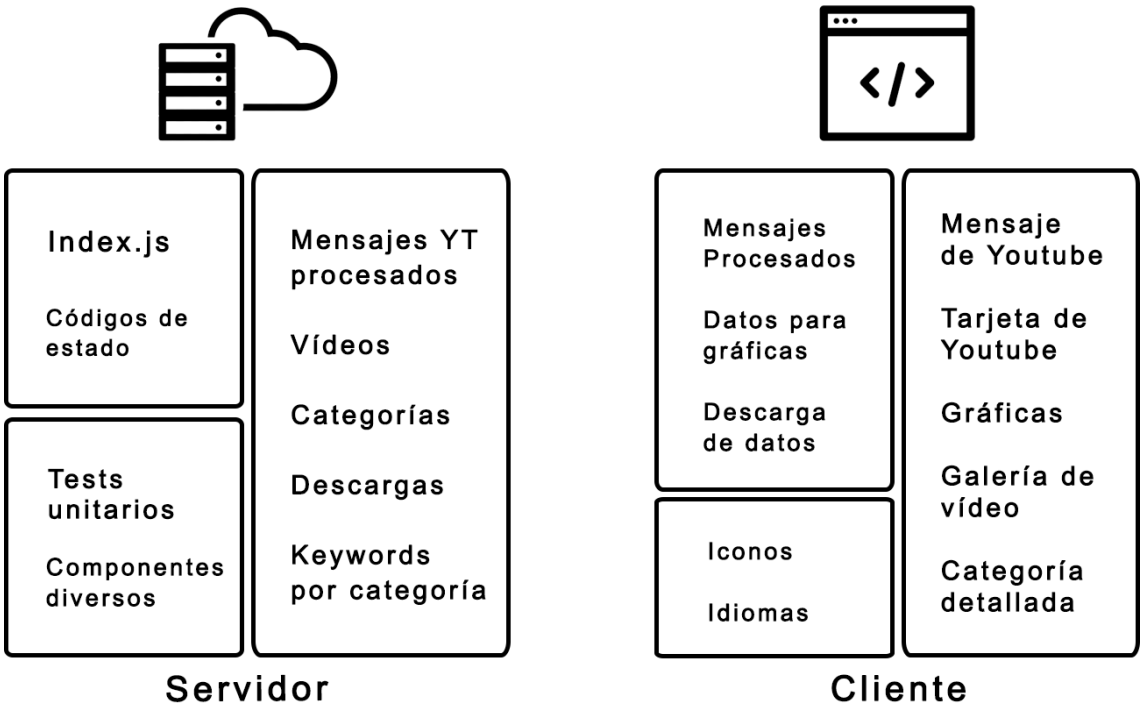

Figura 5.1: Componentes de la aplicación creados y/o modificados en la implementación.

En la figura 5.1 tenemos una vista general de los componentes alterados o creados durante la implementación. En la parte del servidor nos encontramos diversos controladores encargados de gestionar las peticiones del lado cliente. Para satisfacer las consultas de los mensajes procesados y el contenido multimedia relacionado se crean componentes específicos y se amplían los que realizan una búsqueda conjunta de todas las fuentes. En cuanto al funcionamiento de la API, el archivo ''index.js'' contiene las peticiones que

atiende y sobre qué ruta, además de contener los datos de Swagger. Por último, también se realizan modificaciones en algunas clases utilizadas para el correcto funcionamiento de los componentes mencionados anteriormente, como puede ser las declarar variables relativas a YouTube, y la creación de un test unitario para comprobar el funcionamiento de la API. En lo referente a la vista de la aplicación, nos encontramos con la visualización de la respuesta de las peticiones de los mensajes procesados de YouTube, la tarjeta asociada a la fuente, la visualización de las gráficas y la galería de vídeo. Por otro lado, el componente diseñado para visualizar los mensajes procesados de Youtube y la ampliación de los controladores para la descarga de datos y la visualización de gráficas. Por último, la adición de nuevos iconos y frases en los dos idiomas para los que está disponible la aplicación.

5.1 Prototipado

Para el prototipado de la aplicación se ha elegido el software de Figma [18], el cual permite crear diseños de alta fidelidad, y, al estar basado en la nube, resulta de gran utilidad para trabajar de forma conjunta, facilitando la colaboración de los integrantes del equipo y la comunicación durante el proceso. Los prototipos de alta fidelidad dan unas aproximaciones más detalladas al diseño final, y permiten además una evaluación más formal. Con el software de Figma podemos reutilizar parte del prototipado ya que esta herramienta permite copiar como CSS o SVG los elementos web creados, permitiendo añadirlos elementos directamente dentro de la página web. Dado que la metodología seguida consigue una comunicación fluida entre las partes implicadas en el desarrollo, la posible desventaja de utilizar un prototipado de alta fidelidad, que es crear falsas expectativas o una idea equivocada del avance del proyecto, se disipa dado que se tiene acceso a éste. Por lo tanto, tenemos los beneficios de obtener un prototipo de alta fidelidad de forma rápida donde podemos reutilizar parte del prototipado utilizando el CSS asociado.

Dado que la implementación se desarrolla sobre un trabajo ya existente, principalmente se siguen las líneas de diseño ya adoptadas por la aplicación con la intención de que la nueva implementación no se distinga de la anterior. Se mantiene un diseño minimalista que utiliza los espacios y franjas para delimitar distintas áreas dentro de la vista de la aplicación. Esto limita las posibilidades del diseño y acelera el proceso. También se siguen técnicas de diseño responsivo para que la visualización de la página sea adaptable al tamaño y resolución del dispositivo en el que se visualiza la página web. Para el diseño se utiliza Material UI, que permite adaptar la página de manera responsiva de forma nativa. Dado que este diseño adaptable se basa en la división del espacio en doce columnas, al igual que en Bootstrap, supone un beneficio para la división de los espacios con facilidad, pero en este caso concreto del prototipado es también una dificultad. Esto se debe a que, en la vista en la que se visualizan los mensajes procesados, antes de la implementación nos encontramos con cuatro tarjetas: una tarjeta global que muestra todos los mensajes procesados, y una tarjeta para cada red social de la que se extraen los mensajes procesados. Al añadir una nueva tarjeta la división de este espacio se complica. Dado que la tarjeta global muestra las estadísticas de todas las fuentes, en el prototipado se opta por mantener el diseño de las cuatro tarjetas por fila, colocando la tarjeta global en una fila única encima de las otras tarjetas, como se puede observar en la figura 5.2.

Respecto a los mensajes procesados, cada uno es representado dentro de un elemento dentro de una lista. En el borde izquierdo del componente se utiliza el color rojo, "#FF4500", que se utiliza para representar la fuente de YouTube, también utilizado en el icono visible en la parte superior derecha del contenedor. En el otro lateral, al igual que en los elementos de las demás fuentes, se añade el logo de la fuente, en este caso el

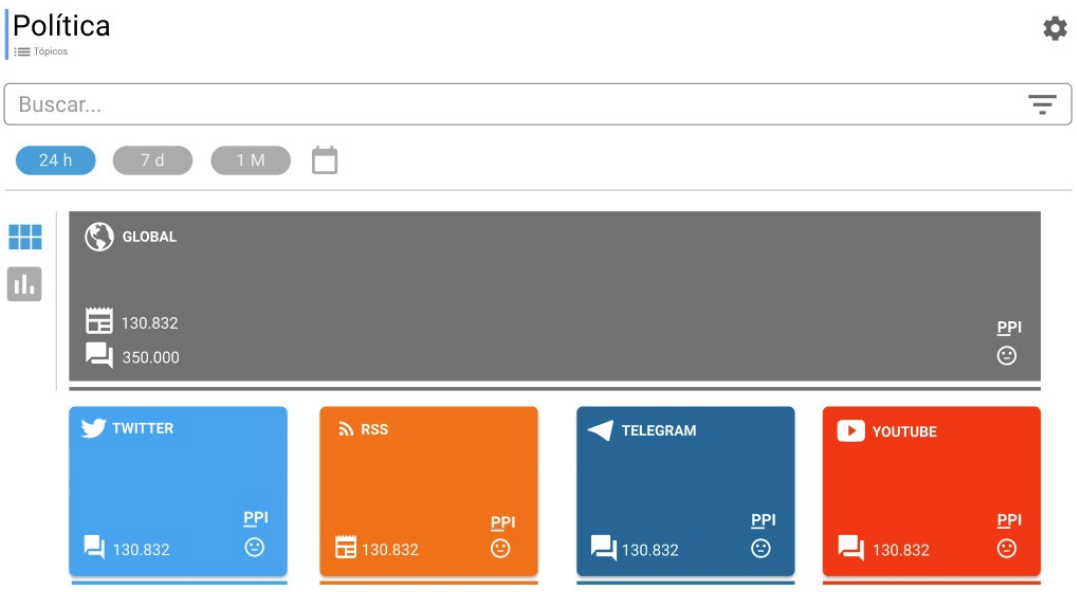

Figura 5.2: Prototipado de la aplicación con la inclusión de Youtube como nueva fuente.

de Youtube, con el mismo color de fondo. Dentro de cada elemento se muestra el título del vídeo en negrita, y debajo de éste la descripción del vídeo, de donde se sacarán las entidades en el procesamiento del mensaje. Para acceder al recurso procesado, al final de la descripción del vídeo, que dependiendo de la longitud mensaje puede estar incompleto ya que la longitud del mensaje está limitada, se añade un enlace con en el que se puede acceder al recurso. Por último, en la parte inferior de cada elemento, se encuentran las entidades y etiquetas encontradas en el lado izquierdo, y en el lado derecho el PPI (percepción del problema) junto a un icono de una cara, que cambia de color en función del sentimiento asociado, y la fecha del mensaje procesado. El prototipado de este componente se puede observar en la figura 5.3.

Figura 5.3: Prototipado de un mensaje procesado.

En el caso de estar seleccionada la tarjeta de YouTube, sólo se visualizan los mensajes procesados de Youtube, teniendo todos los mensajes el aspecto visible en la figura 5.3, mientras que al renderizarse la vista por defecto o si se volviese a escoger después de seleccionar una fuente, la vista de la categoría con sus mensajes procesados tiene la apariencia de la figura 5.4.

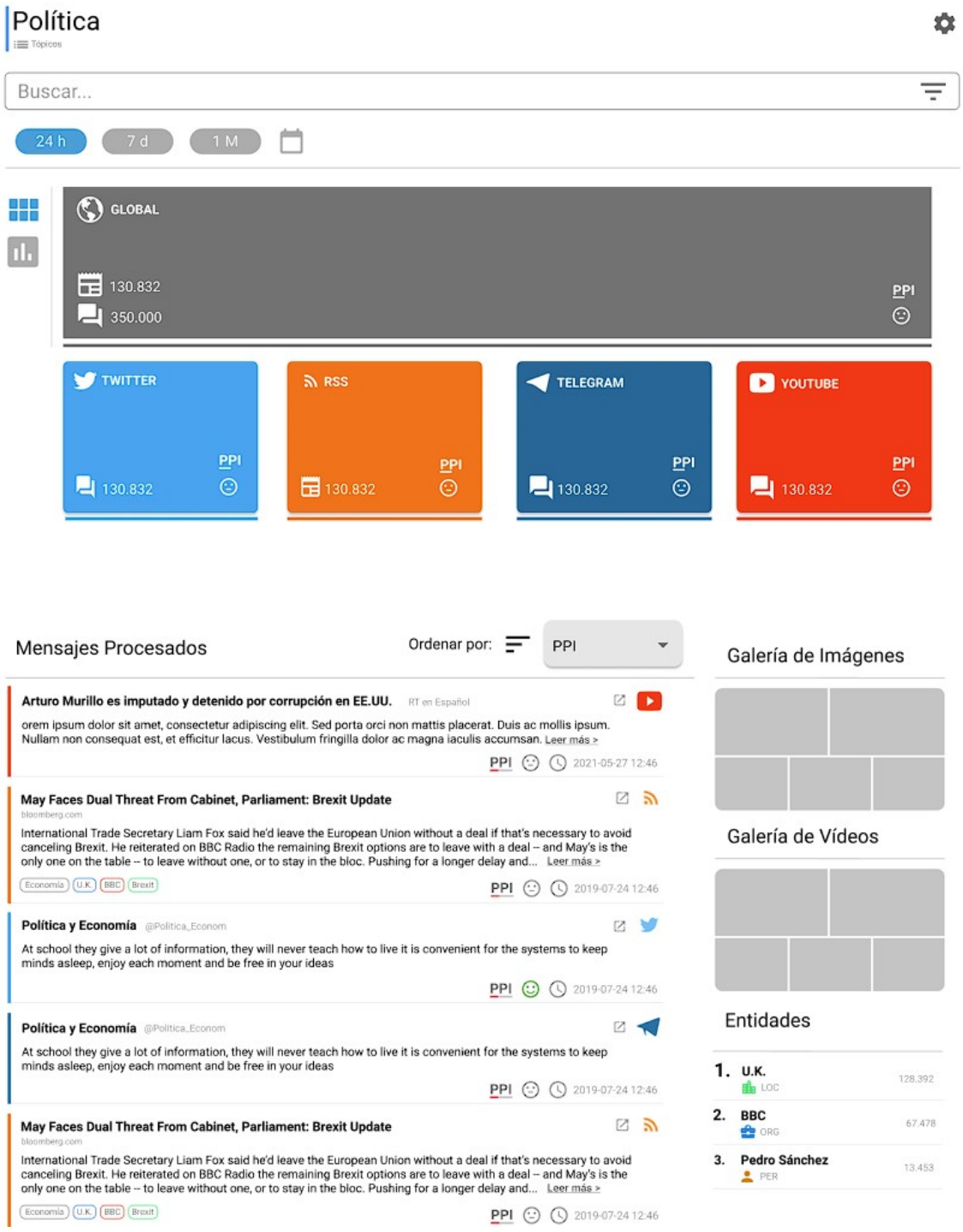

Figura 5.4: Vista general del prototipado.

5.2 Diseño de la solución

En este apartado se detalla el diseño seguido durante el desarrollo de la solución y sus aspectos más relevantes. Tanto la parte *front-end* como la *back-end* utilizan la tecnología JavaScript con sus respectivas librerías. Utilizar un mismo lenguaje de programación facilita el desarrollo al tener en un mismo lenguaje en ambas partes, lo que ayuda a las comunicaciones, además de ofrecer un despliegue más rápido y una amplia variedad de librerías disponibles gracias al uso de NPM, tanto en la parte del servidor como en la del cliente. Además, la popularidad de JavaScript se mantiene como una de las más utilizadas y populares durante los años, mientras que el framework React ha conseguido superar a Angular en numero de usuarios [19], así que se utilizan tecnologías con una gran respaldo de la comunidad detrás, asegurando su continuidad y mejoras.

5.2.1. Front-end

En la parte frontal de la aplicación, al utilizarse React, se han de crear componentes que serán renderizados en tiempo de ejecución a medida que el usuario interactúe con la página. Al tratarse de una ampliación a una aplicación ya creada, además de crear nuevos componentes, se debe ampliar la implementación de componentes ya creados. Para la solución del *front-end* se encuentran tres grupos de elementos a solucionar.

En primer lugar, la gestión de las peticiones al servidor. Como se explica en el capítulo 4.2, para la API creada en el *back-end*, cada red social tiene desarrollado un controlador que se encarga de consultar en el índice de Elasticsearch. Además, todas las fuentes comparten el controlador de la galería de imágenes y el controlador de la galería de vídeos. En el caso de Youtube, se amplia el controlador de la galería de vídeos para que consulte en el índice de Elasticsearch correspondiente. Para ello, deben crearse peticiones para traer los datos siguiendo el diseño REST utilizado por la aplicación, y los parámetros necesarios deben ser extraídos de la página web, en concreto del estado de la vista de React. Nos encontramos con tres tipos de peticiones a realizar:

1. Peticiones para solicitar los mensajes procesados, que deben mostrarse en cada categoría. En este tipo nos encontramos con que, siguiendo el diseño ya existente en la aplicación, se hacen dos peticiones.

   a) Una petición para recuperar los comentarios procesados sin agregaciones.

   b) Una petición para recuperar los comentarios procesados con agregaciones.

2. Peticiones para solicitar los datos que deben incluirse en las gráficas de comparación entre los sensores. La aplicación utiliza sensores físicos y sensores sociales. En el periodo del desarrollo de este trabajo, el sensor físico recoge las estadísticas de casos por Covid-19, que los contrasta con la afluencia de mensajes de las distintas categorías que están creadas en la página.

3. Peticiones para solicitar la descarga de los mensajes procesados de cada categoría. Esta descarga está limitada a 10.000 mensajes, y sólo puede proceder de una fuente de las disponibles, estando disponible en formato JSON o CSV.

En segundo lugar, la creación de elementos dedicados en exclusiva en lo referente a la nueva fuente de datos. Dentro de esta categoría nos encontramos la inclusión del icono de Youtube, la tarjeta de Youtube, y los mensajes procesados de esta fuente. Además del icono utilizado en la tarjeta y en los comentarios para representar la fuente, dentro de la aplicación también nos encontramos con que, para distinguir cada tipo de mensajes,

se utiliza un icono distinto, con el que se representa el número de mensajes procesados de esa fuente, y no la fuente en sí. Estos iconos se extraen, al igual que los demás iconos de la aplicación, de la librería de Material UI, que proporciona más del 1000 iconos para utilizar en las aplicaciones. Respecto a la tarjeta de Youtube, se reutiliza el componente utilizado para renderizar las demás tarjetas de la aplicación: el componente "SocialCard". Éste actúa como contenedor para renderizar la tarjeta, donde recibe como parámetros un identificador, unos iconos, el número de mensajes procesados, y los datos relativos al sentimiento asociado y su percepción general. También nos encontramos con la creación de peticiones dedicadas a la consulta de los datos relativos a Youtube en el servidor.

Por último, actualizar las clases afectadas por la inclusión de una nueva fuente de datos. Esta inclusión implica la revisión de gran parte del código de la aplicación existente, ya que cada componente en el que estén declaradas las fuentes y/o se utilicen, se debe añadir esta nueva junto a las ya existentes. Esto comprende la mayoría de las vistas comunes, como implementar la visualización del número de mensajes procesados de cada categoría, o la vista de los comentarios procesados en una categoría. En esta vista es posible visualizar los comentarios procesados de una categoría, visualizar los gráficos, o descargar los comentarios, por lo que a su vez tiene diferentes vistas que también deben ser actualizadas. Esta categoría es con diferencia una de las más delicadas de la solución, ya que se debe tener un conocimiento completo del funcionamiento del código ya existente, realizando las modificaciones pertinentes para realizar la solución sin crear conflictos con el código ya escrito, siguiendo las mismas reglas para crear variables y métodos para facilitar el entendimiento a los demás programadores involucrados en el desarrollo, tal y como se detalla en el capítulo 7.

5.2.2. Back-end

En el lado del servidor de la aplicación se crea el *endpoint* encargado de recibir las peticiones relativas a Youtube y se actualizan los demás controladores afectados por la inclusión de una nueva fuente. Por lo tanto, nos encontramos con dos categorías dentro del diseño de la solución *back-end* :

1. Controlador encargado de procesar las peticiones tipo 1 y 2 del front-end. Al igual que el resto de componentes, realizan la consulta en Elasticsearch en función de los parámetros obtenidos en la búsqueda.

2. Controladores afectados por la inclusión de la nueva fuente:

    - Controlador ”index.js”, archivo principal en el que están declarados los *endpoint*s creados en el servidor. Al igual que todos los *endpoint*s, el creado también se documenta con el uso de Swagger.
    - Controlador "downloadController.js", donde se añade la comprobación de si la petición enviada requiere descargar los datos de Youtube. En el caso de ser afirmativo, determinar si se requieren en JSON o en CSV, crear el archivo, y devolverlo como respuesta.
    - Controlador "videosController.js", que devuelve los vídeos asociados a cada categoría, que son mostrados en la galería de vídeo de cada vista de la categoría. En el caso de Youtube, disponemos de la fuente del vídeo, y, dado que es una red social donde el contenido relevante es audiovisual, es necesario enviar los datos necesarios para la correcta visualización del vídeo en la galería. También existe un controlador encargado de enviar las imágenes asociadas a una categoría, pero la decisión tomada fue no incluir las miniaturas de los vídeos en la galería, dado que todos los vídeos disponen de una miniatura y

esto podría plagar la galería rápidamente, aumentando en exceso la cantidad de imágenes. Otro motivo es que en Youtube existe una técnica muy utilizada conocida como ”clickbait”, que consiste en generar miniaturas que deforman la realidad o excesivamente llamativas, con la intención de conseguir visitas. El uso de esta técnica podría llegar a confundir al usuario de la página ya que podría encontrarse con imágenes muy alarmistas o directamente falsas.

- Controlador ”wordCountController”. Este controlador devuelve en forma de mapa el conteo de las palabras más repetidas en las búsquedas realizadas en las fuentes. Dado que la petición ya devuelve la búsqueda realizada en las demás fuentes, se sigue el diseño de la solución ya existente y se realiza la búsqueda, añadiendo en la respuesta los resultados de Youtube también.

- Controlador ”categoryDataController”. Controlador encargado de devolver las categorías creadas en la aplicación. Dado que se incluye una nueva fuente se ha de revisar y actualizar todo lo referente a las fuentes para que Youtube sea visible como las demás.

- Controladores ”tempCalcCalls” y ”tempCalls”. Relacionados con el controlador ”catagoryDataController”, devuelven datos y estadísticas de la categoría.

5.3 Detalles de implementación

Dado que el proyecto se ha desarrollado mediante la metodología ágil SCRUM, es conveniente estructurar los detalles de la implementación en base a los sprints realizados. Además de organizar el contenido siguiendo la metodología impartida, identifica si se han cumplido las tareas asignadas en cada sprint.

5.3.1. Sprint 0

En este sprint inicial se realiza la instalación del software requerido para el proyecto, sus dependencias y se comprueba que el funcionamiento de las ramas habilitadas dentro del proyecto de desarrollo funcionan correctamente. Esta fase es la más corta del proyecto y sirve para conocer las tecnologías utilizadas tanto en el *back-end* como en el front-end. No existen tareas definidas, pero dentro de los proyectos es recomendable familiarizarse con el entorno para ser más productivo en los siguientes sprints.

Para instalar el software necesario, con NPM se pueden resolver la mayoría de dependencias relativas al desarrollo, gracias al contenido de los ficheros "package.json"dentro de los proyectos. Estos contienen las librerías utilizadas dentro de la aplicación, y sólo es necesario tener NPM instalado en el sistema operativo. Si está instalado, situándose en el directorio del proyecto, con la siguiente orden se instalan todas las dependencias:

```
npm install package.json
```

Como ejemplo, el contenido del archivo package.json del *front-end* es el visible en la figura 5.5:

```
{
  "name": "front-end",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@carto/carto-vl": "^1.4.6",
    "@carto/mapbox-gl": "^0.48.0-carto1",
    "@formatjs/intl-pluralrules": "^1.5.9",
    "@formatjs/intl-relativetimeformat": "^4.5.16",
    "@material-ui/core": "^4.11.4",
    "@material-ui/icons": "^4.11.2",
    "@material-ui/styles": "^4.11.4",
    "dateformat": "^4.5.1",
    "firebase": "^8.4.3",
    "highcharts": "^7.2.2",
    "highcharts-react-official": "^2.2.2",
    "js-file-download": "^0.4.12",
    "mapbox-gl": "^1.13.1",
    "notistack": "^0.8.9",
    "prop-types": "^15.7.2",
    "react": "^16.14.0",
    "react-color": "^2.19.3",
    "react-dates": "^21.8.0",
    "react-dom": "^16.14.0",
    "react-images": "^0.5.19",
    "react-intl": "^3.12.1",
    "react-mapbox-gl": "^4.8.6",
    "react-player": "^2.9.0",
    "react-router-dom": "^5.2.0",
    "react-scripts": "^4.0.3",
    "socket.io-client": "^2.4.0",
    "universal-cookie": "^4.0.4"
  },
  "scripts": {
    "analyze": "source-map-explorer 'build/static/js/*.js'",
    "preanalyze": "react-scripts build",
    "dev": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  }
}
```


Figura 5.5: Contenido del archivo package.json

5.3.2. Sprint 1

Durante la realización de este sprint se realizan las siguientes tareas:

- Realizar con Figma el prototipado de la versión de escritorio y móvil de la aplicación: una versión de la vista de los mensajes de Youtube procesados y otra global de todos los mensajes.
- Investigar el código proporcionado en el front-end, entender la función de cada componente para determinar cuáles serán necesarios crear y cuáles modificar.
- Crear los componentes que funcionarán como contenedores, identificados durante el análisis del código, identificando las variables y parámetros necesarios para cada uno con el fin de que en la integración con el resto de los componentes sea funcional.

Durante la realización del prototipado surgen ideas que podrían mejorar estéticamente la apariencia de la aplicación, se sugieren dos cambios en la interfaz que pueden ayudar a que la experiencia de uso de la aplicación mejore visualmente:

1. Cambiar el color utilizado para representar la fuente Telegram de "#0088CC" a "#006599". El parecido con el color de Twitter es bastante similar y eso puede dificultar su distinción en pantallas de menor rango de representación de colores, así como posibles filtros que se añaden de forma temporal y automática a veces por los sistemas operativos de los usuarios: el modo de lectura o de luz nocturna, que modifica la temperatura del color de la pantalla. Oscurecer el color utilizado para Telegram podría resolver o minimizar estos efectos. Sin embargo, dentro de la aplicación, dentro de cada mensaje procesado, a la derecha se puede encontrar el icono de la fuente por lo que esta modificación es opcional y no altera la aplicación.

2. Cambiar el color de la franja inferior horizontal que aparece cuando se selecciona una fuente en concreto. Se utiliza el color utilizado tanto en el icono de la aplicación en la parte superior de la página como dentro de la aplicación para escoger la temporalidad. La propuesta es totalmente estética dado que la fuente seleccionada, al estar la franja horizontal debajo, llama la atención del usuario e indica correctamente la fuente utilizada. Utilizando el color de la tarjeta seleccionada se obtiene un resultado más cohesionado estéticamente. Esto es visible en la figura 5.6, donde se observa cómo cada línea horizontal tiene el mismo color que el de la tarjeta.

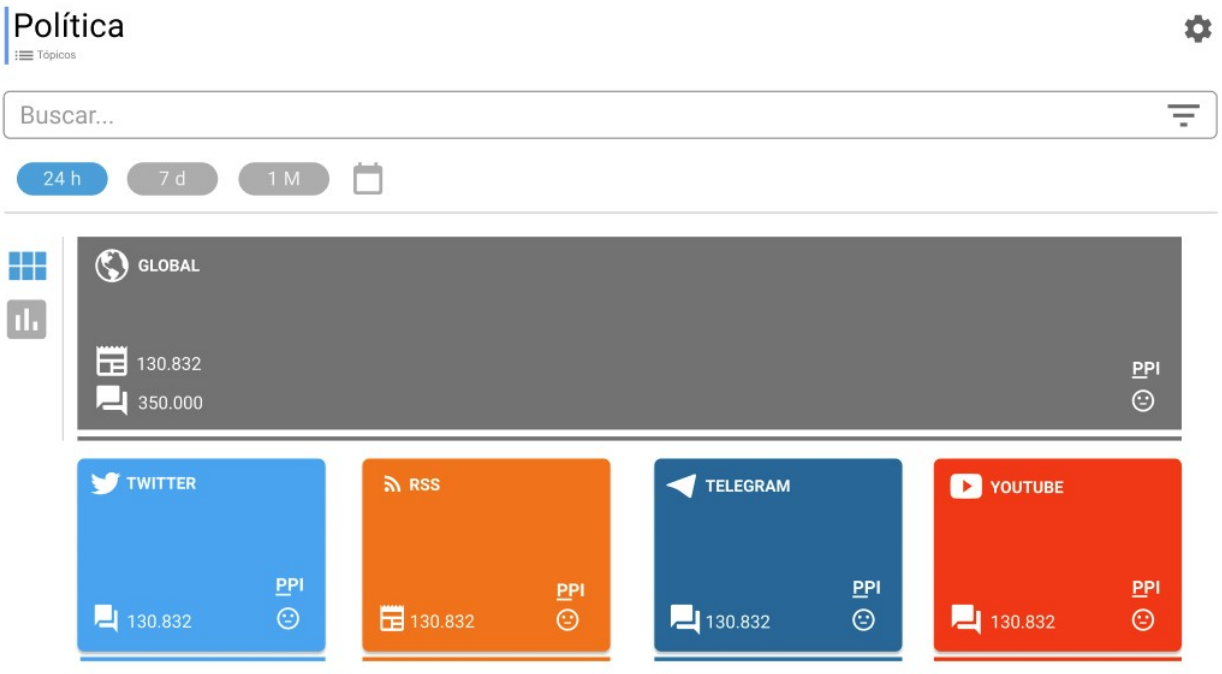

Figura 5.6: Propuestas para la mejora estética de la aplicación

En la reunión del final del sprint, las propuestas de mejora se aceptan junto al prototipado. Una vez terminada la fase del prototipado, se prosigue con la creación de componentes React.

5.3.3. Sprint 2

Una vez creados los componentes que se añaden en la vista de los mensajes procesados de cada categoría, en este segundo sprint se realizan las siguientes tareas:

- Crear una maqueta de los datos que se recibirán en la aplicación en formato JSON para la visualización de los mensajes procesados.
- Utilizar los componentes creados en el sprint anterior para visualizar los datos de la maqueta de los datos en formato JSON. Se utiliza una vista de los mensajes procesados en la que la única fuente disponible es Youtube para evitar conflictos y problemas en el estado de la aplicación.
- Añadir en los ficheros de los idiomas en los que está disponible en el momento del desarrollo de la la aplicación las frases correspondientes. Ya que se trata de una aplicación que está disponible en castellano y en inglés pese a que los mensajes procesados sean en castellano, se deben configurar los mensajes en ambos idiomas en la aplicación.

En este segundo sprint continúa el desarrollo en la parte cliente de la aplicación, centrándose en la visualización de los mensajes procesados. Para acelerar el desarrollo en las fases tempranas del proyecto, se decide utilizar el modelo de los datos que se recibirán en las peticiones para realizar la implementación en la parte del *front-end* antes de implementar el *endpoint* en el *back-end* . Dado que los datos que se reciben son en formato JSON y siempre se espera recibir los mismos contenidos en la respuesta a excepción de que se produzca un fallo en la aplicación, se puede maquetar un JSON para comprobar que la visualización es correcta, y una vez creado el *endpoint* en la parte del servidor se maneje la petición a él. Para evitar interferencias con los componentes ya desarrollados, se utiliza una vista en la que sólo los mensajes procesados de YouTube son visualizados. En posteriores sprints, una de las tareas deberá ser integrar estos componentes con la vista global de los mensajes.

La maqueta del JSON sigue el siguiente formato de la figura 5.7:

Una vez creada la maqueta de los datos que van a visualizarse en la aplicación, se utilizan los componentes creados en el sprint 1. Para ello, los componentes React se basan en los propios objetos disponibles en la librería para crear contenedores de contenido. En relación con la tarjeta de Youtube, el componente que renderiza el contenido es llamado "socialCard". Para poder acceder a los atributos de estado del componente, en React se desestructura la variable del estado:

```
const { classes } = props;
```

En el apartado 8.1 del documento está el código relevante en la construcción del componente que genera la tarjeta de Youtube y las de las demás redes sociales, así como la representación de las variables procesadas por la aplicación, el total de los mensajes procesados de cada red y sus iconos. Como se puede apreciar en el anexo, en la construcción del elemento "Grid", que envuelve el contenido de los mensajes procesados y los iconos, se le proporciona el estilo mediante las propiedades del componente con el atributo "className". Dado que esta tarjeta es utilizada para todas las fuentes, se deber determinar a qué fuente pertenece cada elemento proporcionado para la correcta renderización. Por ello, con el uso de la función *map* se va iterando sobre los datos proporcionados. Para el icono, se determina el tipo en función del valor "processedIcon.icon", obtenido por parámetros. Respecto al total de mensajes procesados, se construye para cada categoría de mensajes

```
"category_id": "5f7651541a0d0621ac64b6c1",
"keywords": [
  {
    "word": "españa",
    "polarity": 0,
    "pos": ""
  },
  {
    "word": "murcia",
    "polarity": 0,
    "pos": ""
  }
],
"video": {
  "region_code": "ES",
  "url": "https://www.youtube.com/watch?v=2OdsQ2jrP6I",
  "channel": {
    "name": "Confirmado La Moncloa",
    "url": "https://www.youtube.com/channel/UCyHC-NMCet0BTtKWsusAZaw"
  },
  "title": "El techo de gasto en España",
  "description": "Hoy martes de octubre comenzamos con una reflexión...",
  "publish_timestamp": 1602021612000
},
"nlp": {
  "category_weight": 0.9999999999999998,
  "sentiment": 0,
  "ppi": 0,
  "entities": [
    {
      "text": "España",
      "label": "LOC",
      "coord": {
        "lat": 39.3262345,
        "lon": -4.8380649,
      }
    },
    {
      "text": "España",
      "label": "LOC",
      "coord": {
        "lat": 39.3262345,
        "lon": -4.8380649,
      }
    }
  ],
  "language": "es"
}
```


Figura 5.7: JSON de un mensaje procesado

un icono junto a la afluencia de los mensajes procesados, donde sólo se renderiza la categoría correcta mediante la comprobación del valor ”processedIcon.typography”. Para otros elementos como las barras horizontales encargadas de medir el impacto del mensaje, o la fecha, el componente se actualiza para que sea compatible con los mensajes procesados de Youtube también creando las variables necesarias para que el contenido de la fuente sea distinguido del resto de las demás. Cuando una de las redes sociales ha sido seleccionada, aparece en su parte inferior una línea del mismo color de la categoría. Para este fin, se realiza una comprobación de cuál es la fuente escogida mediante el uso de la variable ”selected”, disponible en las propiedades React de la aplicación.

Respecto al componente encargado de visualizar los mensajes de la red social, se crea ”commentListItem”. En el apartado 8.2 de este documento se encuentra el código relevante relativo al componente. Profundizando en la implementación, se utiliza el objeto ”ListItem” junto a ”ListItemText” para encapsular el contenido del mensaje. En su propiedad ”primary” se proporciona el título del vídeo de Youtube, valor almacenado en la variable ”processedCommentTitle”, y se repite con la descripción del vídeo, almacenado en ”processedCommentDescription”. Para acceder al recurso de forma externa y poder

acceder al contenido recogido por el crawler, a continuación de la descripción del vídeo se añade un enlace a éste encapsulado en una etiqueta HTML, que llama a una función encargada de abrir en una nueva ventana la URL del vídeo. En la propiedad "secondary" de "ListItemText" se renderiza el contenido relacionado a las etiquetas encontradas y el procesamiento natural del lenguaje, además del icono de YouTube. El icono de Youtube es proporcionado por el paquete de iconos "Material Icons", de Material-UI [20]. Además del icono nos encontramos con un contenedor que una vez comprobado que la variable que recoge las entidades contiene alguna, muestra las entidades encontradas en el texto. Dado que las entidades se almacenan en un vector en cada mensaje procesado, mediante la función *map* se recorre el vector para procesar cada entidad almacenada. Para diferenciar entre los tres tipos de entidades determinadas en la aplicación, se escoge un color para cada tipo de entidad, localización, persona y organización; dejando un color neutro gris en el caso de no haberse determinado de qué tipo es en el procesamiento del mensaje. A continuación, se maneja el contenido relativo al impacto del mensaje. Para ello, se utilizan barras horizontales de progreso, que en función del impacto tienen el progreso más o menos lleno. Aquí, igual que en el enlace a la fuente del vídeo, podemos observar cómo se referencian las frases de los diccionarios de la aplicación, ya que está disponible en dos idiomas. Por último, se muestra la fecha del mensaje, que se calcula con la marca temporal disponible en los datos del mensaje procesado. Esta marca temporal se manda como argumento a una función ya diseñada por la aplicación, utilizada para determinar la fecha de los mensajes procesados de las diversas fuentes.

En cuanto a los idiomas, la aplicación tiene la interfaz disponible tanto en castellano como en inglés. Gracias al componente ya implementado "languageSwitch", en la aplicación se encuentra la posibilidad de cambiar de idioma. Cualquier texto que no sea de mensajes procesados está reflejado en un documento en el que se recogen todas las frases utilizadas en la aplicación siguiendo un formato concreto. Esta solución permite cambiar el idioma de la aplicación de manera dinámica ya que cara mensaje queda guardado e identificado, por lo que en los componentes no se escribe el mensaje, sino la referencia al mensaje dentro del documento. El formato seguido es el nombre del componente en el que se localiza la frase, la variable asociada al mensaje si existiese y luego su identificador. Como podemos ver en el siguiente ejemplo, frase que se muestra indicando el número de mensajes de Youtube procesados en la clase "categoryCard" seguiría el siguiente formato:

```
''categoryCard.processedYoutubeMessages'': ''Mensajes de Youtube procesados''
```

5.3.4. Sprint 3

Gracias a la instalación en el sprint 0 del proyecto del *back-end* , en este tercer sprint se puede comenzar con la implementación del *endpoint* encargado de realizar la consulta a Elasticsearch. Las tareas realizadas en este sprint son las siguientes:

- Analizar el funcionamiento de las consultas en Elasticsearch y los tipos de búsquedas que se pueden realizar.
- Crear las consultas a Elasticsearch necesarias para obtener los mensajes procesados de la categoría que se solicite en la petición.
- Crear las respuestas que devuelve el *endpoint* al recibir peticiones:
  - Peticiones de los mensajes procesados sobre una o más categorías.
  - Peticiones de la afluencia de mensajes para comparar el tráfico con los sensores físicos.

- Gestionar la comunicación del *endpoint*, enviando correctamente los datos.
- Gestionar la recepción de los datos en el *front-end* gestionando la recepción de los datos.

Respecto a las consultas en Elasticsearch, los datos se organizan en índices. Estos índices que se encuentran en format JSON. Para realizar la búsqueda se sigue la siguiente sintaxis:

```
URL + "/" + índice + "/" + tipo + "/" + "_search"
```

Donde la URL es la dirección donde se aloja Elasticsearch, el índice, en el que se encuentran los mensajes procesados es "youtube-search"; el tipo del documento es "_doc" y la búsqueda se efectúa con "_search". Una de las grandes ventajas de Elastic a la hora de realizar las búsquedas es que además de aceptar búsquedas complejas con los tipos definidos, también es compatible con las búsquedas del tipo SQL, por lo que podrían realizarse de esta manera y sería compatible. Esta característica ayuda a familiarizarse con las búsquedas. Como el resto de las búsquedas hechas en la aplicación están hechas de un mismo modo, para mantener la cohesión del código realizado y facilitar el trabajo en equipo, se ha seguido el mismo estilo de búsqueda con JSONs.

Los parámetros obligatorios que se requieren en la búsqueda de los mensajes procesados son los siguientes:

- **CategoryId**: es el identificador único de cada categoría. Utilizado para filtrar el contenido del índice y recibir sólo los mensajes procesados pertenecientes a ésta. Todos los identificadores tienen una longitud de 24 caracteres alfanuméricos.
- **OldestTimestamp - newestTimestamp**: estos dos parámetros funcionan de forma conjunta para determinar el rango temporal en el que se debe realizar la consulta. OldestTimestamp indica la marca temporal más antigua que satisface la búsqueda, mientras que newestTimestamp el más reciente, que por lo general suele ser el momento exacto de la petición. El intervalo de tiempo es determinado desde el cliente, aunque puede ser personalizable desde un calendario, existen por defecto los valores de una hora, un día, una semana; además de un mes en la vista de las categorías.
- **From**: utilizado para determinar la cantidad de mensajes procesados que se están visualizando en la aplicación y tener un control del flujo de estos. En resumen, actúa como puntero al último mensaje procesado, para que el servidor procese la consulta y devuelva los siguientes mensajes a partir del indicado.
- **Size**: este parámetro se utiliza para determinar la cantidad de mensajes procesados que se le pide al controlador. De forma fija se establece que se pidan 10 mensajes nuevos procesados para evitar sobrecargas.

En lo relativo a los datos que se visualizan en los gráficos, los parámetros son únicamente **categoryId**, **oldestTimestamp** y **newestTimestamp**. En el anexo 8.3 se puede encontrar un ejemplo de la búsqueda en el nodo de Elastic.

Para la gestión de errores, antes de realizar la búsqueda se comprueba si los identificadores recibidos en la consulta satisfacen las características para que devuelva resultados. Dado que cada identificador tiene una longitud de 24 caracteres, la primera acción del controlador es comprobar que las categorías tienen dicha longitud. En el caso de no satisfacerse, no se realizaría la consulta y se devuelve un código de estado 400 con un mensaje indicando que no hay datos disponibles:

```
let errorCategories = 0;
let categoryIds = req.params.categoryIds.split(",");
let categoryIdQuery = [];
categoryIds.forEach(categoryId => {
  if (categoryId.length == 24) {
    categoryIdQuery.push({
      "term": {
        "category_id": categoryId
      }
    })
  } else {
      errorCategories++;
  }
})

if (errorCategories > 0) {
    return res.status(httpStatusCodes.BAD_REQUEST).send({
      errMessage: "There is no data available"
    });
}
```

Figura 5.8: Comprobación de las categorías de una petición HTTP.

Una vez creadas las consultas, es necesario devolverlas siguiendo el formato esperado dentro de la aplicación. En el sprint 2 se detalla el formato a seguir de los datos de los mensajes procesados, por lo que la construcción de esta consulta debe devolver los datos de la misma forma. Además de los mensajes procesados, para una categoría tenemos diversos datos que son requeridos en la aplicación, como el número total de mensajes, que es la cantidad respuestas de la consulta de Elastic que satisface la petición; la media del sentimiento asociado o el impacto de los mensajes procesados. Para ello, dentro del JSON tenemos diversas primitivas que contienen los siguientes datos:

- **hits**: número de aciertos de la búsqueda.
- **data**: vector que contiene los mensajes procesados.
- **avg**: estadísticas de los mensajes procesados: el sentimiento asociado, el impacto del problema, la percepción del problema y el impacto del mensaje.
- **timeSeriesCount**: frecuencia de aparición de los mensajes.
- **locationEntities**: número de apariciones de una entidad y ubicación aproximada.

La respuesta creada para la petición de los mensajes con agregaciones y esperada por el *front-end* sigue el siguiente formato seguido en la figura 5.9. La versión sin agregaciones sólo tiene los campos ”hits” y ”data”.

```
const response = {
    "hits": body.hits.total.value,
    "data": hits,
    "avg": {
      "sentiment": body.aggregations.avg_sentiment.value,
      "ppi": body.aggregations.avg_ppi.value,
      "pii": body.aggregations.avg_pii.value,
      "tii": body.aggregations.avg_tii.value
    },
    "timeSeriesCount": timeSeriesCount,
```

Figura 5.9: JSON que se envía como respuesta a una petición HTTP.

Respecto al formato de los mensajes de cada categoría para su contraste con los sensores físicos, las demás fuentes siguen el siguiente formato y ha sido respetado:

```
{
    "timeSeriesCountById": {
        "{categoryId1}": [
            {
                "timestamp": 1468800000,
                "weight": 13,
                "highPpiWeight": 2
            },
            {
                "timestamp": 1555900000,
                "weight": 1352,
                "highPpiWeight": 628
            }
        ],
        "{categoryId2}": [
            {
                "timestamp": 1858800000,
                "weight": 14,
                "highPpiWeight": 5
            },
            {
                "timestamp": 1556500000,
                "weight": 1112,
                "highPpiWeight": 84
            }
        ]
    }
}
```


Figura 5.10: JSON de respuesta para la representación en gráficos.

Donde cada *categoryId* es el identificador asignado a cada categoría, y cada categoría tiene tres valores siempre: la marca temporal, el peso y el impacto del problema.

Para que el servidor procese estas peticiones, en el archivo *index.js*, que contiene todas las peticiones y contiene los datos visibles en el swagger, deben estar presentes las peticiones, por lo que se añaden junto a las ya existentes:

```
//Mensajes procesados
router.route('/youtube/:categoryIds/:oldestTimestamp-:newestTimestamp')
  .get(processedYoutubeCommentController.getHistoricCommentsValidations,
      processedYoutubeCommentController.getHistoricComments)

//Datos para las graficas
router.route('/youtube/chart-data/:categoryIds/:oldestTimestamp-:newestTimestamp')
  .get(processedYoutubeCommentController.getChartData)
```

Una vez tenemos las peticiones gestionadas en el servidor, la recepción en la parte cliente se realiza con las funciones creadas en la clase "youtubeRequests.js", explicadas en el sprint 3. Para recibir el la respuesta de las peticiones correctamente, se utiliza el objeto **promise**, utilizado para las comunicaciones asíncronas en JavaScript. En caso de recibir una respuesta, se comprueba que sea un objeto y se procede a actualizar el estado de la aplicación. En caso de una respuesta negativa, se comunica dentro de la aplicación mediante una notificación emergente.

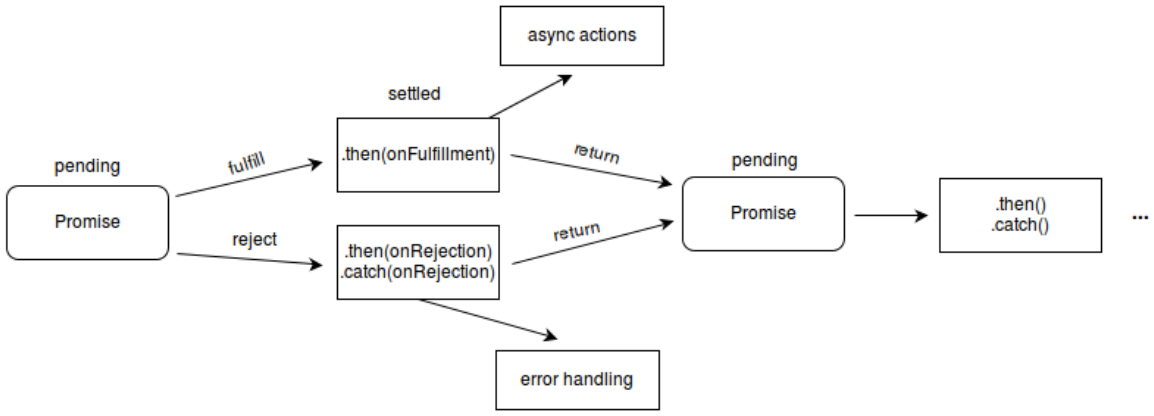

Figura 5.11: Flujo de ejecución de una promesa en JavaScript
Fuente: https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Promise

5.3.5. Sprint 4

El cuarto sprint se enfoca en añadir el resultado de las tareas de anteriores sprints dentro de la aplicación. Las tareas realizadas son:

- Añadir los componentes creados en anteriores sprints en el componente "detailedCategoryTopicsAndAuthors.js", encargado de renderizar la vista de los mensajes procesados de una categoría; además de ampliar los componentes que utiliza para el renderizado de la aplicación, donde se encuentra la gran parte de la ampliación de este trabajo en la parte del *front-end*.
- Actualizar las variables de estado de React de forma que se reflejen los mensajes procesados de Youtube.
- Actualizar en el *back-end* el controlador encargado de la búsqueda de vídeos en los mensajes procesados para que se recojan también los vídeos de Youtube.

Durante el transcurso de este sprint se encuentra gran parte de la implementación del *front-end* en el proyecto. El componente "detailedCategoryTopicsAndAuthors.js", más que un componente actúa como prácticamente una vista entera a excepción del navegador superior de la aplicación. En esta clase nos encontramos la mayoría de los componentes afectados por la inclusión de esta nueva fuente. Además de los componentes afectados para la visualización de los mensajes procesados de las diversas fuentes, existen funciones dentro del componente encargadas de las peticiones a la API, la recepción de imágenes y vídeos, el manejo de las descargas de vídeo, la visualización de las gráficas y el desplazamiento vertical infinito –prácticamente infinito, hasta que no haya más mensajes disponibles– dentro de la visualización de los mensajes procesados.

En relación a los mensajes procesados nos encontramos con la inclusión de los componentes creados en el sprint como son la tarjeta de la nueva fuente, que muestra el número de mensajes procesados y funciona a su vez para visualizar sólo los mensajes de esa fuente al clicar en la tarjeta; y el componente creado utilizado en la lista de los mensajes procesados, donde cada mensaje procesado es enviado a través de la aplicación para que sea renderizado con este componente. Además, se aprovecha para realizar la modificación del color asociado a Telegram para que se distinga mejor de Twitter. También se incluyen las funciones encargadas de realizar la petición de los mensajes procesados de Youtube y su recepción, desarrollado en el sprint 3. No obstante, dentro del componente que incluye todas las fuentes, existen otras variables que deben ser modificadas y actualizadas al recibir los mensajes procesados de Youtube, por lo que se realiza una revisión de

todas las variables implicadas en el proceso de la visualización de los mensajes. La gran mayoría de estas se encuentran en ”state”, la variable que contiene todas las propiedades del estado del componente que se está renderizando en tiempo de ejecución. Respecto a estas variables, las más relevantes son:

- socialNetworks[]: vector que contiene las redes sociales de la categoría, Twitter, noticias, Telegram y Youtube, y el conjunto de estas.
- tabActive: variable de estado utilizada para alternar entre las tres pestañas de la vista, visualización de mensajes, visualización de gráficas y descarga de datos. Por defecto está inicializada para mostrar la pestaña de los mensajes procesados.
- socialNetwork: variable de estado utilizada para determinar la red social seleccionada. Por defecto se inicializa con el valor "global"para que se visualicen los mensajes de todas las fuentes.
- processedYoutubeComments[]: vector de estado que recoge los mensajes de Youtube procesados. Se actualiza al recibir la respuesta del servidor.
- youtubeLocationEntities[]: vector de estado que almacena las localizaciones encontradas en el procesamiento de los mensajes de Youtube.
- loadingHistoricYoutubeComments: variable booleana de estado que se utiliza para determinar si se está esperando la respuesta del servidor. Cuando su valor es verdadero, se muestra una barra horizontal en la interfaz para comunicar al usuario que el contenido se está cargado.
- youtube: variable de estado que representa la red social y sus atributos. Entre los atributos podemos encontrar el número total de mensajes, la media de los valores del procesamiento natural del lenguaje o el número de mensajes procesados.
- totalYoutubeComments: variable de estado que almacena el número de mensajes procesados.
- hasMoreProcessedYoutubeComments: variable booleana de estado que determina si se pueden seguir mandando peticiones al servidor o si ya no existen más resultados de búsqueda. Al recibir la petición del servidor se compara el valor del número total de mensajes de la red social con el almacenado en *totalYoutubeComments*, en el caso de ser mayor el número de mensajes totales de los ya procesados, su valor es verdadero y se permite seguir realizando peticiones para recuperar mensajes de la red social.
- infiniteScrollLoadingYoutubeComments: variable utilizada para manejar el desplazamiento infinito en la aplicación. Técnicamente no es infinito y existe un momento en el que la aplicación puede quedarse sin mensajes, pero el volumen de los mensajes procesados es elevado.
- Vectores utilizados para los gráficos: se utilizan vectores dedicados a almacenar el contenido que se muestra en cada tipo de gráficos.

Respecto a la visualización de mensajes, dentro de la vista de las categorías hay un apartado en específico para la representación visual de los mensajes, su afluencia, sentimiento y localización. Cada vista de una categoría nos encontramos con los siguientes tipos de representaciones:

1. Gráfico continuo horizontal. Este tipo de gráfico representa la afluencia de los mensajes en función del tiempo. Utiliza el componente *streamGraph* que recibe como parámetros *streamGraphSeries*, el vector se almacena la cantidad de mensajes de cada red social; *streamGraphTimeSeries*, vector de afluencia de mensajes en función del tiempo; y el vector *colors*, que contiene los colores asociados a cada red social para representar en el gráfico con los mismos colores la afluencia de los mensajes. Los componentes reciben los datos en las peticiones al servidor, actualizándose en la recepción del JSON. con este componente se puede observar la evolución de la afluencia de mensajes de las diversas redes sociales de manera más gráfica y permite al usuario determinar la importancia de la categoría o las categorías escogidas en las diversas fuentes de datos. Poniendo un ejemplo, en el tema de la vacunación se puede observar cómo la principal fuente es Twitter mientras que en Telegram se encuentra esta categoría mucho menos.

2. Volumen de mensajes y su sentimiento. Este gráfico representa los mensajes procesados en función del tiempo también, pero en este se refleja el sentimiento asociado al tema o temas escogidos, dejando un poco de lado la fuente de los mensajes. Esta representación nos permite saber la percepción de los usuarios respecto al tema escogido. Utiliza el componente *SentimentVolumeBySNChart* y recibe como parámetros los punteros *stackedBarSeries* y *stackedBarCategories*, y los datos introducidos se reciben en la petición al servidor. El sentimiento asociado a la categoría nos permite saber si dentro de la afluencia de mensajes, la categoría tiene un sentimiento más positivo o negativo, o simplemente neutral. Esta gráfica combina muy bien con la primera, ya que podemos conocer cuánto se habla de un tema, y si es percibido como algo negativo o positivo; ya que mensajes con una alta positividad o negatividad, sin tener en cuenta la afluencia de los mensajes no tiene tanto peso.

3. Conteo de palabras. En esta gráfica se muestran las palabras más repetidas dentro de una categoría, identificadas en función de la fuente. El componente encargado de renderizar esta nube de palabras es *wordCloudChart*, recibiendo como parámetros el vector *wordCount*, que contiene las palabras más relevantes de la categoría. Se actualiza cuando se recibe la respuesta del servidor tras la petición del contenido. Este componente nos permite visualizar localizaciones, fechas o entidades relevantes dentro de una categoría y ver la importancia respecto a las demás, ya que cuanto más aparece una palabra más grande es representada.

4. Representación basada en la localización. Dado que los mensajes procesados disponen de una localización aproximada, la representación en un mapa puede ser de gran utilidad para ver en qué zonas está siendo más relevante una categoría, como por ejemplo una guerra o un terremoto. Como mapa se utiliza el proporcionado por OpenStreetMap, donde se puede visualizar la afluencia de mensajes por localización y fuente. En el mapa, según la afluencia de mensajes en cada localización, se genera un puntero que contiene los mensajes procesados que provienen de esa localización. Si existen mensajes procesados en la vista, el componente *locationMap* es encargado de la renderización, que recibe como parámetros las variables los vectores de entidades asociadas a cada categoría en cada red social.

Cada uno de los componentes mencionados dispone de una leyenda que permite activar o desactivar la visualización de la fuente en el gráfico, por lo que se permite filtrar por la fuente en cualquier momento y en cualquiera de las gráficas. En las variables asociadas a la leyenda se añade la nueva fuente y se rediseña el tamaño de ésta, ya que tenía un tamaño ajustado para tres fuentes, por lo que el estilo del componente se modifica para hacerlo más ancho por defecto y que las cuatro fuentes se visualicen en una única línea en vez de en dos, ocupando más espacio.

En cuanto a la descarga de los datos, se crea una nueva pestaña en la vista correspondiente para la descarga de los mensajes procesados de Youtube. Como en las otras tres fuentes ya presentes, se crea un ejemplo de datos para que el usuario pueda conocer la información relativa a los mensajes que puede obtener. La descarga de los mensajes, disponible en formato CSV o JSON, requiere una petición al controlador encargado de la API. Este controlador es ampliado para poder gestionar la descarga de los mensajes procesados de Youtube. Para ello se realiza una consulta en el índice de Elasticsearch y se maneja la creación del archivo CSV o JSON para entregarlo en la respuesta a la petición. Esta búsqueda en Elastic recibe los mismos parámetros que si se tratase de una búsqueda de mensajes para visualizarlos, pero la cantidad de los mensajes es superior.

Como finalmente no se añaden las miniaturas de los vídeos ya que la galería puede llevar a engaño sobre el contenido por las miniaturas "clickbait" de los creadores de Youtube, la parte relacionada con la galería de vídeo queda asignada en el quinto sprint. La causa de esto es un mejor reparto de la carga de trabajo entre sprints, ya que la actualización de la lógica encargada de recibir y visualizar la galería de vídeos no supone una gran complejidad en la parte frontal de la aplicación.

5.3.6. Sprint 5

En el quinto sprint se termina la implementación relativa a la visualización de los mensajes procesados de Youtube. Las tareas realizadas son:

- Actualizar el componente encargado del conteo de palabras, *categoryWordCountsController*, para que procese también los mensajes de Youtube y devuelva el su resultado al *front-end*.
- Actualizar todas las tarjetas que representan las categorías de la aplicación para que se visualice el número mensajes procesados de Youtube.
- Ampliar el controlador encargado de generar los ficheros en formato JSON de los mensajes procesados para que también maneje las peticiones de los mensajes procesados de YouTube.
- Ampliar el controlador encargado de generar los ficheros en formato CSV de los mensajes procesados para que también maneje las peticiones de los mensajes procesados de YouTube.

El controlador del conteo de palabras realiza una búsqueda en los índices de Elasticsearch, obteniendo los parámetros de la consulta mediante los parámetros petición GET gestionada, de forma similiar a los mensajes procesados. Por lo tanto, se realiza una consulta para cada red social. Una vez obtenidos los resultados de las consultas, para cada red social se obtienen las palabras más repetidas en la categoría escogida además de su afluencia. Como respuesta, se devuelve un objeto JSON en el que están las palabras más repetidas en cada red social. El desarrollo en este componente consiste en realizar la búsqueda en Elasticsearch junto a las demás y preparar el JSON de respuesta, añadiendo un nuevo objeto JSON junto a los de las redes sociales ya existentes. En la parte *front-end* están creadas variables necesarias para visualizar en el mapa de palabras estos datos.

Los datos visualizados por tarjetas que representan las categorías de la aplicación están manejados por un controlador en específico, encargado de consultar en Elasticsearch el número de mensajes disponibles para cada categoría. En este escenario se repite lo descrito en el controlador del conteo de palabras, creando las variables necesarias para que se almacenen los datos de Youtube y creando una nueva consulta en el índice dedicado a los mensajes procesados de Youtube.

Respecto a la descarga de los datos, existe un único controlador que gestiona la descarga de datos. Este controlador filtra si el contenido que debe generarse es un objeto JSON o CSV mediante los parámetros recibidos en la petición GET. Como se detalla en el capítulo 5.2.2, no se permite que el número de mensajes exceda de diez mil, ya que es muy recomendable determinar algún límite de la aplicación para evitar errores o colapsos. Independientemente del formato escogido para la descarga, se sigue un formato único para representar los mensajes obtenidos. El esquema de datos es:

```
youtubeCommentKeyMap = [
  "video.title",
  "video.description",
  "video.channel.name",
  "video.url",
  "video.publish_timestamp",
  "nlp.language",
  "nlp.ppi",
  "nlp.sentiment",
  "nlp.category_weight",
  "nlp.entities.text",
  "nlp.entities.label",
  "nlp.entities.coord.alt",
  "nlp.entities.coord.lat",
  "nlp.entities.coord.lon"
]
```

Figura 5.12: Esquema de los datos disponibles para descargar.

5.3.7. Sprint 6

En el último sprint se realizan las pruebas y la comprobación de que la aplicación funciona correctamente y tiene el comportamiento esperado. En el capítulo 6 se detallan los resultados de la validación y las prueba. Las tareas identificadas en este sprint son:

- Desarrollar un test unitario encargado de comprobar la gestión de las peticiones al controlador creado.
- Utilizar el paquete *express-validator* comprobar que los parámetros pasados al controlador por el cliente satisfacen los requisitos identificados.

Para la realización de las pruebas del controlador encargado de las búsquedas de Youtube se utiliza Chai [21] junto a Mocha.JS [22] para el entorno de pruebas, ambas librerías disponibles para Node.JS e instalables mediante NPM. Chai es una biblioteca de aserciones que permite la realización de estos tests basándose de dos tipos de metodologías, el desarrollo guiado por pruebas (TDD) y el desarrollo guiado por comportamiento (BDD). En este caso se utilizan las aserciones del estilo "should", que describen el comportamiento que debería tener la aplicación dadas unas circunstancias. Cada una de las aserciones está constituida por una cadena de sentencias fácilmente legibles para el usuario.

En las pruebas diseñadas en Chai se expresan los comportamientos que se esperan en la aplicación bajo una situación. Se realiza un test unitario para comprobar el funcionamiento del componente creado en la API y el manejo de excepciones en la ejecución. Como se detalla en la figura 4.3, el controlador devuelve dos códigos de estado. Para ello, en el código de la prueba se le envían peticiones correctas e incorrectas. Para concentrar todas estas pruebas en un único fichero, se crea un archivo llamado ”youtube-requests.js” que contiene todas las pruebas descritas a continuación.

Como el entorno de pruebas y las pruebas se realizan en el entorno de desarrollo, la aplicación se ejecuta de forma local. Para reflejarlo en el test unitario se declara una variable común en todas las pruebas que apunta a la dirección local del ordenador. Las librerías de Chai utilizadas son "chai-http" y "chai-json". Se importan en el código del test:

```
chai.use(require('chai-http'));
chai.use(require('chai-json'));
```

En primer lugar, los mensajes procesados, dada una categoría y un intervalo de tiempo se debe obtener como respuesta un objeto JSON y un código de estado 200 si la petición se ha realizado correctamente. Se le proporcionan datos necesarios al test para que la aplicación responda correctamente:

```
it('returns category', (done) => {
  chai.request(url)
  .get('/api/youtube/5f7651541a0d0621ac64b6c1/1624316515802-1624921315802')
  .end( function(err,res){
    console.log(res.body)
    expect(res.body).to.be.a.jsonObj();
    expect(res).to.have.status(200);
    done();
  });
});
```

Figura 5.13: Comprobación de una petición correcta de mensajes procesados.

En el caso de estar enviándose una categoría no existente, la aplicación debe devolver un código de estado 400:

```
it('returns 400 bad request', (done) => {
  chai.request(url)
  .get('/api/youtube/5f7651541a0d06214b6c1/1624316515802-1624921315802')
  .end(function (err, res) {
    console.log(res.status)
    expect(res).to.have.status(400);
    done();
  })
});
```

Figura 5.14: Comprobación de una petición incorrecta de mensajes procesados.

En segundo lugar, nos encontramos con un comportamiento similar en las peticiones de los datos para su contraste con los sensores físicos. Dadas unas categorías existentes, la respuesta del servidor también debe ser un objeto JSON y el código de estado 200:

```
it('returns charts data', (done) => {
  chai.request(url)
  .get('/api/youtube/chart-data/5f6a40914512ed002d577109,5f7651541a0d0621ac64b6c1,
  5f92c9ad9c35b62bc67d72e6,5fb4259162ea851156e98a46/1624365375717-1624970175717?
  timeSeriesInterval=1d')
  .end( function(err,res){
    console.log(res.body)
    expect(res.body).to.be.a.jsonObj();
    expect(res).to.have.status(200);
    done();
  });
});
```

Figura 5.15: Petición correcta de los datos para la visualización en los gráficos.

Si por el contrario no se proporcionasen unas categorías correctas, sea porque no existen o porque la longitud del identificador de la categoría no es correcta, la respuesta debe ser un código de estado 400, ya que así se gestiona en el controlador:

```
it('returns 400 bad request', (done) => {
  chai.request(url)
  .get('/api/youtube/chart-data/5f6a414512ed002d577109,5f7651541a0d0623341ac64b6c1,
  5f92c9ac35b623bc67d72e6,5fb4233359162ea851156e98a46/1624365375717-1624970175717
  ?timeSeriesInterval=1d')
  .end(function (err, res) {
    console.log(res.status)
    expect(res).to.have.status(400);
    done();
  })
});
```

Figura 5.16: Petición incorrecta de los datos para la visualización en los gráficos.

Una vez asegurado el tipo de las variables con Express-validator, para realizar el test en Chai se utiliza el entorno de pruebas de Mocha, añadiendo en *package.json* un nuevo script para iniciar las pruebas del archivo "youtube-requests.js" con Chai:

```
"youtube-requests.js": "mocha"
```

Respecto a la validación de los parámetros proporcionados en la petición, para su comprobación se utiliza Express-Validator [23]. Captura los parámetros GET de las peticiones y comprueba los requisitos especificados en la validación, que deben cumplirse para que se realice la petición. Así como Chai, ofrece un uso similar al lenguaje natural que permite crear unas comprobaciones fácilmente legibles. La validación de los parámetros se basa en comprobar que son del tipo esperado y se realiza como puede observarse en la figura 5.17.

```
exports.getHistoricCommentsValidations = [
  check('categoryId')
    .exists().withMessage('id must be in url params')
    .isString().withMessage('id must be a string'),
  check('newestTimestamp')
    .exists().withMessage('newestTimestamp must be in url params')
    .isString().withMessage('newestTimestamp must be of type string'),
  check('oldestTimestamp')
    .exists().withMessage('oldestTimestamp must be in url params')
    .isString().withMessage('oldestTimestamp must be of type string'),
  check('from')
    .isString().withMessage('from must be a string'),
  check('size')
    .isString().withMessage('size must be a string'),
  check('aggregations')
    .isBoolean().withMessage('aggregations must be a boolean'),
  check('order')
    .isString().withMessage('order must be a string'),
  check('orderType')
    .isString().withMessage('orderType must be a string'),
  check('timeSeriesInterval')
    .isString().withMessage('timeSeriesInterval must be a string'),
     check('selectedText')
    .isArray().withMessage('selectedText must be a string'),
  check('selectedEntities')
    .isArray().withMessage('selectedEntities must be an array'),
  check('selectedKeywords')
    .isArray().withMessage('selectedKeywords must be an array'),
]
```

Figura 5.17: Validación de los parámetros de las peticiones mediante Express-Validator.

CAPÍTULO 6
Validación y pruebas

Dado que la metodología utilizada durante el proyecto ha sido SCRUM, se ha seguido un modelo de trabajo iterativo. Es por eso que en todos los sprints se han hecho comprobaciones del funcionamiento de los componentes desarrollados para así confirmar la finalización de las tareas. A pesar de la frecuente revisión, realizar las pruebas una vez finalizada la aplicación es una tarea necesaria para finalizar un proyecto. En este capítulo se exponen los resultados obtenidos en los tests unitarios creados en el sprint 6 y las demás herramientas utilizadas para la evaluación del sistema.

Respecto al resultado de los tests, las pruebas ejecutan consiguen ser superadas. Los tests de los mensajes procesados responden lo esperado. En la figura 6.1 se observa el uso de Chai además de la información dada sobre las peticiones recibidas en el servidor. La primera, que da unos datos correctos, da la información de la petición, como la ruta, la versión de HTTP empleada y el código de estado. En el caso de una petición mal realizada, se muestra información relativa a la petición:

```
  GET category
range: 1624921315 , 1624316515
------------------------------------------------------------------------
------------------------------------------------------------------------
info: ::ffff:127.0.0.1 - - [01/Jul/2021:09:15:19 +0000] "GET /api/youtube/5f7651541a0d0621ac64b6c1/1624316515802-1624921315802 HTTP/1.1" 200 11752 - 153.091

    ✓ returns category (176ms)
range: 1624921315 , 1624316515
------------------------------------------------------------------------
------------------------------------------------------------------------
error: ::ffff:127.0.0.1 - - [01/Jul/2021:09:15:19 +0000] "GET /api/youtube/5f7651541a0d06214b6c1/1624316515802-1624921315802 HTTP/1.1" 400 43 - 2.596

    ✓ returns with bad request
```

Figura 6.1: Tests de los mensajes procesados.

Los datos de las gráficas también se resuelven satisfactoriamente, viendo la misma información sobre las peticiones cuando se realizan:

```
  GET charts
[ { term: { category_id: '5f6a40914512ed002d577109' } },
  { term: { category_id: '5f7651541a0d0621ac64b6c1' } },
  { term: { category_id: '5f92c9ad9c35b62bc67d72e6' } },
  { term: { category_id: '5fb4259162ea851156e98a46' } } ]
info: ::ffff:127.0.0.1 - - [01/Jul/2021:09:15:19 +0000] "GET /api/youtube/chart-data/5f6a40914512ed002d577109,5f7651541a0d0621ac64b6c1,5f92c9ad9c35b62bc67d72e
6,5fb4259162ea851156e98a46/1624365375717-1624970175717?timeSeriesInterval=1d HTTP/1.1" 200 755 - 95.789

    ✓ returns charts data (101ms)
[]
error: ::ffff:127.0.0.1 - - [01/Jul/2021:09:15:19 +0000] "GET /api/youtube/chart-data/5f6a414512ed002d577109,5f7651541a0d0623341ac64b6c1,5f92c9ac35b623bc67d72
e6,5fb4233359162ea851156e98a46/1624365375717-1624970175717?timeSeriesInterval=1d HTTP/1.1" 400 43 - 0.782

    ✓ returns with bad request


  4 passing (297ms)
```

Figura 6.2: Tests de los datos para las gráficas.

En consecuencia, los controladores implementados en el servidor responden satisfactoriamente a las peticiones y manejan las excepciones que se puedan producir en el caso de realizarse una petición con parámetros mal indicados.

Para la evaluación de la parte *front-end*, realizada por el equipo de desarrollo, se han tenido en cuenta los siguientes aspectos [24]:

- ¿Los objetivos del sitio web son concretos y reconocibles? ¿Los contenidos ofrecidos corresponden con los objetivos?

  La aplicación web muestra los temas más hablados en internet y las preocupaciones de la población, mostrando los mensajes más relevantes e impactantes junto al contenido multimedia relacionado; además de comparar la afluencia de los mensajes recogidos con sensores físicos. Por lo tanto, sí son concretos y reconocibles y el contenido ofrecido corresponde a los objetivos.

- ¿Tiene una URL correcta, clara y fácil de recordar? ¿Y las URL de las páginas internas?

  El dominio de la aplicación es corto y conciso. Sin embargo las URL de las páginas internas, pese a seguir la estructura de un diseño REST, para las categorías e intervalos de tiempo se utilizan identificadores de gran longitud alfanuméricos. Esto crea URLs internas de una gran longitud que no proporcionan una información clara de la categoría visualizada.

- ¿Se evita la redundancia de enlaces?

  Sí, para acceder a una vista o recurso la aplicación web está diseñada para se acceda desde un único lugar.

- ¿El sitio web se actualiza periódicamente?

  Sí, el sitio web por defecto muestra los mensajes procesados más recientemente. Esta ordenación puede modificarse dentro de cada categoría en la aplicación.

- ¿Se emplea un lenguaje claro y conciso?

  Sí, En la aplicación se evita el uso de frases extensas. El lenguaje utilizado es claro y accesible para cualquier tipo de usuario. Por ejemplo, en la vista general de las categorías se utilizan preguntas para organizar el contenido, mostrando a continuación de la pregunta el contenido relacionado a la cuestión.

- ¿Los rótulos son significativos?

  Sí, proporcionan información del contenido mostrado en la aplicación.

- ¿Los enlaces son reconocibles?

  Sí. Dentro de la aplicación se permite el acceso a la fuente original de las noticias y los vídeos de Youtube. En ambos casos existe un icono reconocible para acceder al recurso o existe un enlace que se diferencia dentro de la aplicación para que se identifique como un enlace.

- ¿El diseño y la estructura de la aplicación es adecuada y coherente?

  Sí, es adecuada y coherente, sigue los estándares modernos del diseño web con el sistema basado en columnas que permite organizar el contenido de manera dinámica.

- ¿Se hace un uso correcto del espacio visual de la página utilizando zonas "en blanco" entre los objetos para poder descansar la vista?

  Sí, en la aplicación se delimitan los distintos apartados mediante el uso de espacios en blanco y franjas.

- ¿El uso del buscador interno es accesible?

  Sí, dentro de una categoría el buscador es uno de los elementos que se visualizan en primer lugar, ocupando prácticamente todo el ancho de la pantalla.

- ¿El buscador permite la búsqueda avanzada?

  Sí, al realizar una búsqueda se muestran en una lista desplegable palabras clave, localizaciones y organizaciones debajo del cuadro de búsqueda. Están diferenciadas utilizando un color distinto para cada tipo de entidad.

- ¿Asiste al usuario en caso de no ofrecer resultados para una consulta?

  Sí, si se realiza una búsqueda que no ofrece resultados la aplicación contesta indicando que no se han encontrado datos disponibles para el criterio de búsqueda.

- ¿Incluyen los elementos de la web atributos que describan el contenido?

  Sí. Cuando se pasa el ratón por encima de iconos o las estadísticas del procesamiento natural del lenguaje, se muestra información relativa al contenido. Por ejemplo, si se pasa el ratón por encima del icono de un reloj en cada mensaje procesado, aparece un diálogo en el que se muestra "fecha del mensaje"; si se pasa el ratón por encima del número de mensajes procesados de una fuente, aparece un diálogo en el que se muestra un mensaje indicado que es el número de mensajes procesados.

- ¿La aplicación web es compatible con distintos tipos de navegadores?

  Sí, es compatible con Mozilla Firefox, Opera, Safari y los navegadores basados en Chromium.

- ¿La aplicación web se puede visualizar correctamente desde dispositivos de distinto tipo?

  Sí, la aplicación es accesible desde *smartphones*, tabletas y ordenadores de escritorio gracias al la implementación de la aplicación utilizando el diseño responsivo.

- ¿La aplicación web se puede utilizar sin descargar o instalar extensiones o aplicaciones adicionales?

  Sí, dado que utiliza tecnologías completamente compatibles con los navegadores mencionados.

En general, tras la evaluación se encuentra con un resultado positivo, con un diseño responsivo y adaptable a cualquier dispositivo, minimalista y eficaz, que organiza el contenido de forma estructurada y hace un buen uso de los espacios en blanco. Como aspecto mejorable en el aspecto de las URL de las páginas internas, dado que no proporciona información al usuario.

Una vez realizadas las pruebas en ambas partes de la aplicación, en la figura 6.3 podemos observar la aplicación web en funcionamiento en la vista de la visualización de mensajes, escogida la categoría ”agua”. Se aprecia que el número de mensajes procesados por parte de la aplicación es inferior al número de *tweets* o noticias. Están visibles las En los mensajes procesados en los que se han detectado entidades, en el caso del primer mensaje visible, dos localizaciones: Lanza, un concello de Galicia; y Alberguería. Además del componente creado para la visualización de los mensajes procesados, se puede observar cómo se diferencia mejor la fuente de Twitter respecto a la de Telegram por el cambio del color propuesto en la fase del prototipado, y la tarjeta utilizada para representar los mensajes de Youtube.

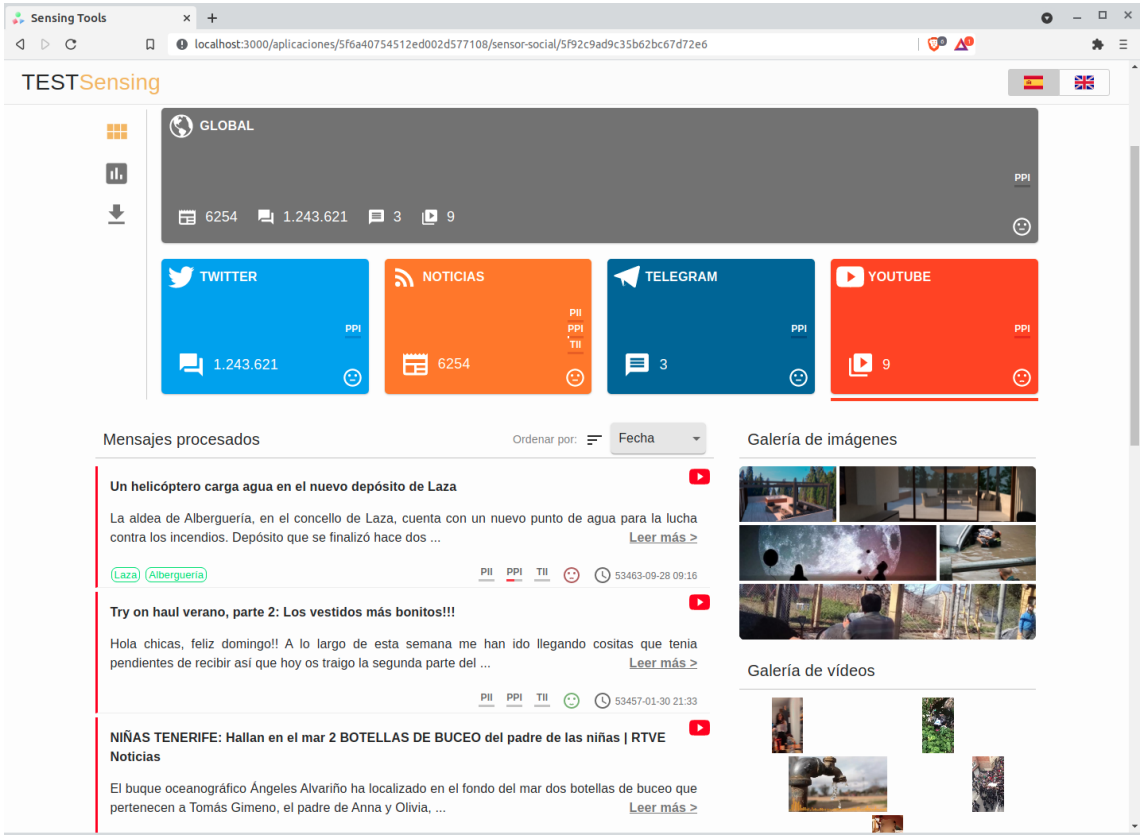

Figura 6.3: Vista de la aplicación en funcionamiento.

CAPÍTULO 7
Conclusiones y trabajos futuros

7.1 Conclusiones

En el desarrollo del proyecto se ha indagado sobre una nueva arquitectura de aplicaciones web, las SPA, y las tecnologías disponibles para su creación. El uso de JavaScript en la parte del servidor y el cliente, salvando las diferencias de las librerías utilizadas, supone una gran ventaja en cuanto al tiempo necesario para el análisis y el desarrollo de la aplicación. En relación a las librerías y el lenguaje de programación utilizado, se ha puesto en evidencia el gran potencial que tiene el desarrollo full stack con JavaScript dada su versatilidad y la gran cantidad de librerías disponibles, desde la renderización de HTML hasta la realización de tests unitarios.

Centrándonos en el desarrollo del lado cliente, se ha descubierto un gran potencial en las librerías React, React Route y Material UI, que permiten la creación de aplicaciones web de forma rápida con un diseño responsivo mucho más eficiente que el uso de HTML, CSS y JavaScript por separado. La formación autodidacta de estas librerías, desconocidas hasta el inicio del proyecto, han ocupado una parte del proyecto que cabe destacar, dado que dista de las tecnologías empleadas en el contenido impartido en las asignaturas dedicadas al desarrollo web. En la parte del servidor, se han podido demostrar satisfactoriamente los conocimientos adquiridos en programación y redes. Sumando todo en conjunto, en general se han adquirido conocimientos del trabajo relativo al desarrollo *full stack*.

Respecto a la gestión del proyecto, el uso de SCRUM ha sido clave para mantener una comunicación más fluida entre el equipo de desarrollo y los demás implicados, además de ser una metodología que permite adaptar el transcurso del proyecto a las situaciones específicas que se dan, y eso resulta de gran utilidad. El desarrollo en equipos, sean más grandes o pequeños, son una realidad en el mercado laboral. Durante el proyecto se ha aprendido sobre las repercusiones reales de trabajar conjuntamente en un proyecto. En primer lugar, ampliar una aplicación ya desarrollada requiere de un gran análisis para realizar una correcta integración de la solución. La metodología ágil escogida, con reuniones frecuentes, ha sido clave para facilitar la comprensión del proyecto ya existente y generar un buen ambiente de trabajo. En segundo lugar, el reparto equilibrado de las tareas en el tiempo es importante para asegurar el éxito del trabajo, y si no se realiza una buena gestión de los recursos del equipo pueden surgir complicaciones que hagan peligrar la realización del trabajo en el periodo establecido. Por último, los beneficios que supone la implementación de buenas prácticas en el desarrollo de código, como documentar el desarrollo y programar de forma que el código adquiera una fácil trazabilidad. Facilitar la lectura del código es clave para obtener un buen ritmo de trabajo, sea en grandes o pequeños proyectos, o incluso de una sola persona.

7.2 Trabajos futuros

Como ampliaciones de la aplicación y el contenido que ofrece, se encuentran dos campos que se podrían seguir explotando. Primero, la recogida de mensajes de distintas redes sociales, por ejemplo, la adición de Instagram o Facebook aumentaría la cantidad de contenido multimedia en la aplicación. Ambas supodrían una mejora de contenido en la web debido al gran número de usuarios que contienen ambas aplicaciones, teniendo en cuenta las posibilidades de recogida de datos que proporcione la aplicación o puedan crearse. Segundo, continuar explotando las posibilidades de representación de los mensajes y sus metadatos, sea creando nuevas gráficas basadas en la localización o las entidades relacionadas, o incluso la relación entre la localización y las entidades.

Por otra parte, también se debería mejorar la seguridad de la aplicación. La lógica de los componentes React, las peticiones y utilidades de la aplicación son visibles si se inspecciona la web con las herramientas de desarrollo. Imposibilitar a posibles atacantes la lectura del código de la aplicación y su funcionamiento es una tarea recomendada en la aplicación, ya que cuanta más información puedan recopilar posibles atacantes, peor.

7.3 Relación con los estudios cursados

El objetivo de la realización de este trabajo es aplicar conocimientos recibidos a lo largo de los estudios del grado con el fin de demostrar que pueden diseñar soluciones a problemas reales. Dado que este proyecto se realiza dentro de una aplicación web, los conocimientos adquiridos las asignaturas relacionadas con las redes, el desarrollo web y la programación han sido la base fundamental para el correcto desarrollo del trabajo, sin olvidar otras asignaturas más enfocadas en las metodologías de trabajo y la gestión de un proyecto. Las asignaturas relacionadas con el trabajo son:

- Ingeniería del software (ISW): Metodologías de trabajo en equipos de desarrollo y arquitectura software.
- Gestión de proyectos (GPR): Planificación temporal y asignación de recursos dentro de un proyecto.
- Desarrollo web (DEW): Diseño y desarrollo *front-end*, manejo de peticiones HTTP.
- Integración de aplicaciones (IAP): APIs REST, comunicaciones asíncronas y flujos de datos.
- Desarrollo centrado en el usuario (DCU): Diseño y validación de interfaces web.

Además, durante la realización del proyecto se han demostrado cualidades ensenciales en el ámbito laboral, las competencias transversales, aplicables a diversas áreas y contextos. Se ha trabajado las siguientes competencias:

- Aplicación y pensamiento práctico.
- Análisis y resolución de problemas.
- Diseño y proyecto.
- Trabajo en equipo.
- Aprendizaje permanente.
- Planificación y gestión del tiempo.

Bibliografía

[1] Dong Wang, Tarek Abdelzaher, and Lance Kaplan. *Social sensing: building reliable systems on unreliable data*. Morgan Kaufmann, 2015.

[2] Sasank Reddy, Katie Shilton, Gleb Denisov, Christian Cenizal, Deborah Estrin, and Mani Srivastava. Biketastic: sensing and mapping for better biking. *CHI '10: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1817—-1820, 2010.

[3] Sasank Reddy, Deborah Estrin, and Mani Srivastava. Recruitment framework for participatory sensing data collections. In *International Conference on Pervasive Computing*, pages 138–155. Springer, 2010.

[4] Bret Hull, Vladimir Bychkovsky, Yang Zhang, Kevin Chen, Michel Goraczko, Allen Miu, Eugene Shih, Hari Balakrishnan, and Samuel Madden. Cartel: a distributed mobile sensor computing system. In *Proceedings of the 4th international conference on Embedded networked sensor systems*, pages 125–138, 2006.

[5] Chunming Gao, Fanyu Kong, and Jindong Tan. Healthaware: Tackling obesity with health aware smart phone systems. In *2009 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 1549–1554. Ieee, 2009.

[6] Shane B Eisenman, Emiliano Miluzzo, Nicholas D Lane, Ronald A Peterson, Gahng-Seop Ahn, and Andrew T Campbell. Bikenet: A mobile sensing system for cyclist experience mapping. *ACM Transactions on Sensor Networks (TOSN)*, 6(1):1–39, 2010.

[7] Xiaobo Tan. Autonomous robotic fish as mobile sensor platforms: Challenges and potential solutions. *Marine Technology Society Journal*, 45(4):31–40, 2011.

[8] Anmol Madan, Sai T Moturu, David Lazer, and Alex Pentland. Social sensing: obesity, unhealthy eating and exercise in face-to-face networks. In *Wireless Health 2010*, pages 104–110. 2010.

[9] Manuel Carlos Díaz-Galiano, Manuel García Vega, Edgar Casasola, Luis Chiruzzo, Miguel Ángel García Cumbreras, Eugenio Martínez Cámara, Daniela Moctezuma, Arturo Montejo-Ráez, Marco Antonio Sobrevilla Cabezudo, Eric Sadit Tellez, et al. Overview of tass 2019: One more further for the global spanish sentiment analysis corpus. In *IberLEF@ SEPLN*, pages 550–560, 2019.

[10] Zelin Wang, Zhijian Wu, Ruimin Wang, and Yafeng Ren. Twitter sarcasm detection exploiting a context-based model. In *international conference on web information systems engineering*, pages 77–91. Springer, 2015.

[11] Bing APIs Team. How to track customer sentiment online with bing news search api and text analytics api. Accessed: June 3, 2021.

[12] Francisco Borrego. ¿ puede existir un verdadero periodismo alternativo en redes sociales?: estructuras virales de poder y estrategias de información a través de perfiles falsos. In *Actas del I Congreso Internacional Comunicación y Pensamiento. Comunicracia y desarrollo social (2016), p 1180-1197*. Egregius, 2016.

[13] Pere Masip, Sue Aran-Ramspott, Carlos Ruiz-Caballero, Jaume Suau, Ester Almenar, and David Puertas-Graell. Onsumo informativo y cobertura mediática durante el confinamiento por el covid-19: sobreinformación, sesgo ideológico y sensacionalismo. *El profesional de la información (EPI)*, 29(3), 2020.

[14] Manuel Jesús Caro Cabrera and Luis Navarro Ardoy. La medición del miedo al delito a través de los barómetros del cis. *Reis: Revista española de investigaciones sociológicas, 157, 23-44.*, 2017.

[15] Jeff Sutherland Ken Schwaber. Scrum guide - https://scrumguides.org/scrum-guide.html. Accessed: May 7, 2021.

[16] NodeJS.org. Acerca de nodejs - https://nodejs.org/es/about/. Accessed: June 12, 2021.

[17] ReactJS.org. Presentando jsx - https://es.reactjs.org/docs/introducing-jsx.html. Accessed: June 12, 2021.

[18] About figma. Design, prototype, and gather feedback all in one place with figma - https://www.figma.com/design/. Accessed: April 4, 2021.

[19] Altexsoft.com. Mean and mern stacks: Full stack javascript development explained - https://www.altexsoft.com/blog/engineering/mean-mern-javascript-full-stack/. Accessed: June 5, 2021.

[20] Material-UI. Material icons api - https://material-ui.com/components/material-icons/. Accessed: May 5, 2021.

[21] Chai Assertion Library. Assertion styles guide - https://www.chaijs.com/guide/styles/. Accessed: June 16, 2021.

[22] Mocha.JS. Mocha's api documentation - https://mochajs.org/api/. Accessed: June 17, 2021.

[23] Express validator. Express validator - check api https://express-validator.github.io/docs/check-api.html. Accessed: June 17, 2021.

[24] Hassan-Montero Yusef and F.J. Fernández. Guía de evaluación heurística de sitios web. *No sólo usabilidad*, 2:1–7, 01 2003. Accessed: June 6, 2021.

CAPÍTULO 8
Anexo

8.1 Componente "socialCard"

El valor que puede tener "props.selected" puede ser: "global", "twitter", "rss", "telegram" o "youtube". Se determina cuál es la fuente seleccionada:

```
let tw = false; let rs = false; let tg = false; let yt = false;
  switch (props.selected) {
    case "twitter":
      tw = true;
      break;
    case "rss":
      rs = true;
      break;
    case "telegram":
      tg = true;
      break;
    case "youtube":
      yt = true;
      break;
    default:
      tw = false;
      rs = false;
      tg = false;
      yt = false;
      break;
  }
```

En la creación del contenedor donde se encuentran los datos de cada red social, se comprueba que la fuente ha sido la seleccionada:

```
      <div className={props.selected === props.id && tw ?
          classes.selectedTwitter :
                    props.selected === props.id && rs ? classes.selectedRSS :
                      props.selected === props.id && tg ?
                          classes.selectedTelegram :
                        props.selected === props.id && yt ?
                            classes.selectedYoutube :
                        props.selected === props.id ? classes.selected :
                                 ""}>
```

Los atributos ”selectedTwitter”, ”selectedRSS”, ”selectedTelegram”, ”selectedYoutube” y ”selected” siguen esta estructura, cambiando el color asociado a cada uno de ellos:

```
selectedYoutube: {
    borderBottom: "4px " + " (\textit{color asociado a la fuente}) " + " solid",
    paddingBottom: 5
  },
```

Renderizado de valores de variables de estado en el componente "socialCard":

```
<Grid container direction="row" justify="flex-start" alignItems="center"
    className={classes.statsIcons}>
  {
    props.processedIcons.map((processedIcon, i) => (
      <TextIcon
        key={i}
        className={classes.textIcon}
        divider={true}
        icon={processedIcon.icon}
        iconSpace={processedIcon.iconSpace}
        tooltip={processedIcon.tooltip}
      >
        {processedIcon.typography === 'totalMessages' && <Typography
            component="span" color="inherit" style={{ display: "inline",
            fontSize: 18 }}>{isNaN(props.stats.totalMessages) ?
            props.stats.totalMessages : <FormattedNumber
            value={props.stats.totalMessages} />}</Typography>}
        {processedIcon.typography === 'totalArticles' && <Typography
            component="span" color="inherit" style={{ display: "inline",
            fontSize: 18 }}>{isNaN(props.stats.totalArticles) ?
            props.stats.totalArticles : <FormattedNumber
            value={props.stats.totalArticles} />}</Typography>}
        {processedIcon.typography === 'totalTelegramMessages' && <Typography
            component="span" color="inherit" style={{ display: "inline",
            fontSize: 18 }}>{isNaN(props.stats.totalTelegramMessages) ?
            props.stats.totalTelegramMessages : <FormattedNumber
            value={props.stats.totalTelegramMessages} />}</Typography>}
        {processedIcon.typography === 'totalYoutubeComments' && <Typography
            component="span" color="inherit" style={{ display: "inline",
            fontSize: 18 }}>{isNaN(props.stats.totalYoutubeComments) ?
            props.stats.totalYoutubeComments : <FormattedNumber
            value={props.stats.totalYoutubeComments} />}</Typography>}
         
      </TextIcon>
    ))
  }
</Grid>
```

8.2 Componente ”commentListItem”

```
return (
    <ListItem alignItems="flex-start" className={classes.comment}>
      <ListItemText
        //Titulo del video y descripcion
        primary={
        <Fragment>
```

```
  <div className={classes.title}>
    {processedCommentTitle}
  </div>
  <div>
      <p className={classes.text}>{processedCommentDescription}
      <span
        className={classes.openInNewIcon}
        onClick={() => this.handleOnClickOpenInNew(videoUrl)}
      >
        <FormattedMessage id="articleListItem.readMore" /> &gt;
      </span>
      </p>
  </div>
</Fragment>
}
//tags y PLN
secondary={
<Fragment>
  <SvgIcon viewBox="0 0 512 512" className={classes.youtubeIcon}>
    <YoutubeIcon />
  </SvgIcon>
  <div className={classes.tagsInfo}>
    {
      processedComment.nlp.entities && (
        <Fragment>
          {processedComment.nlp.entities.length > 0 && <span>
               </span>}
          {
            processedComment.nlp.entities
              .filter((entity, i, self) => i === self.findIndex((e) =>
                  (e.text === entity.text)))
              .filter((entity, i) =>
                  (this.processedCommentText.entityCounter[i] && entity))
              .filter((entity, i) => i < 3)
              .map((entity, i) => (
                <Tag
                  key={i}
                  tagType={"EntityKeyword"}
                  name={entity.text}
                  ocurrences={this.processedCommentText.entityCounter[i]}
                  color={
                    entity.label === "LOC"
                      ? "#00DE66"
                      : entity.label === "PER"
                        ? "#DE8500"
                        : entity.label === "ORG"
                          ? "#0D88E1"
                          : "#777777"
                  }
                />
              )).sort((a, b) => (b.props.ocurrences - a.props.ocurrences))
        }
        </Fragment>
      )}
  </div>

  <div className={classes.commentInfo}>
    <HorizontalProgressBar
      className={classes.graphInfoIcon}
```

```
  tooltip={<span><FormattedMessage
      id="youtubeListItem.commentProblemImpact" />:
      <strong><FormattedNumber value={pii} maximumFractionDigits={5}
      /></strong></span>}
  title={"PII"}
  titleColor={'#6B6B6B'}
  percentage={pii * 100}
  barBackgroundColor={'#C4C4C4'}
  progressBarBackgroundColor={'#f53838'}
  width={'20'}
  height={'42'}
/>
<HorizontalProgressBar
  className={classes.graphInfoIcon}
  tooltip={<span><FormattedMessage
      id="youtubeListItem.problemPerception" />:
      <strong><FormattedNumber value={ppi} maximumFractionDigits={5}
      /></strong></span>}
  title={"PPI"}
  titleColor={'#6B6B6B'}
  percentage={ppi * 100}
  barBackgroundColor={'#C4C4C4'}
  progressBarBackgroundColor={'#f53838'}
  width={'20'}
  height={'42'}
/>
<HorizontalProgressBar
  className={classes.graphInfoIcon}
  tooltip={<span><FormattedMessage
      id="youtubeListItem.youtubePerception" />:
      <strong><FormattedNumber value={tii} maximumFractionDigits={5}
      /></strong></span>}
  title={"TII"}
  titleColor={'#6B6B6B'}
  percentage={tii * 100}
  barBackgroundColor={'#C4C4C4'}
  progressBarBackgroundColor={'#f53838'}
  width={'20'}
  height={'42'}
/>

<TextIcon
  className={classes.textInfoIcon}
  divider={true}
  icon={<SentimentIcon sentiment={sentiment}
      className={classes.infoIcon} />}
  iconSpace={0}
  tooltip={<span><FormattedMessage
      id="youtubeListItem.sentimentIndicator" />:
      <strong><FormattedNumber value={sentiment}
      maximumFractionDigits={2} /></strong></span>}
>
</TextIcon>

<TextIcon
  className={classes.textInfoIcon}
  divider={true}
  icon={<AccessTimeIcon className={classes.infoIcon} />}
  iconSpace={0}
```

```
            tooltip={<span><FormattedMessage id="youtubeListItem.messageDate"
                /> </span>}
          >
            <Typography component="span" color="inherit" style={{ display:
                "inline", fontSize: 12 }}>
                {tsToDateStrMin(processedComment.video.publish_timestamp)}</Typography>
          </TextIcon>

        </div>
        </Fragment>
      }
    />
    </ListItem>
);
```

8.3 Consulta en un índice de Elasticsearch

Un ejemplo de una consulta en Elasticsearch siguiendo el formato JSON sería la que se muestra a continuación. Se efectúa una búsqueda booleana en la que deben cumplirse todos los requisitos que contiene. En este caso, debe tener el identificador de la categoría requerido por parámetros, y además estar dentro del rango determinado por newestTimestamp y oldestTimestamp. Por último, debe cumplir también una ordenación marcada por el parámetro orderType.

```
var elasticsearchQuery = {
   "from": req.query.from,
   "size": req.query.size,
   "track_total_hits": true,
    "query": {
      "bool": {
        "must": [
          {
            "bool": {
              "should": categoryIdQuery
            }
          },
          {
            "range": {
                "video.publish_timestamp": {
                    "lte": newestTimestamp,
                    "gte": oldestTimestamp
                }
            }
          }
        ]
      }
    }
    ,
    "sort": [
      {
        [orderPath]: {
          "order": req.query.orderType
        }
      }
    ]
```

```
}
```